Degree Final Project

Bachelor Degree in Informatics Engineering

Detection of cryptocurrency mining malware from network measurements

16 January 2020

Author: **Arnau Beramendi Higueras**

Director: **Pere Barlet Ros**

Specialization: **Computing**

2019 - 2020 Q1

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contents

List of Figures

1 Abstract

Currently cryptocurrencies play an important role in our society. Their popularity has increased hugely in recent years and, consequently, they have attracted the attention of an important segment of the population, which frequently finds in the mining of these cryptographic currencies a new opportunity to earn money. However, this has brought a new scenario where some people use hijacked resources to mine for their own profit. In this context, it is crucial to detect when a host is infected by malware that mines cryptocurrency without permission. Nowadays, there are some approaches to solve this problem: checking the content of each packet (DPI), blocking connections to known pools, analysing the memory consumption or installing anti malware software. Nevertheless, these previous solutions may be quite expensive in terms of resources and money. Additionally, they may require a significant modification of the network or they may be inaccurate in several cases. For this reason, in this project I suggest three different machine learning algorithms, where each one explores a specific feature of this kind of malware in order to detect it. The first one uses flow measurements, the second one uses the DNS queries to detect connections to pools, and the last one uses again the DNS queries but in order to detect connections to malicious domains that may download miners. The combination of these three algorithms provides a very reliable and efficient system to detect which hosts on the network are mining.

2 Introduction and contextualization

2.1 Context and problem formulation

The concept of distributed electronic cash system appeared for the first time in 1998, when Wei Dai published a description of his "b-money" system, but it was not until 2009 that the first decentralised cryptocurrency, Bitcoin, emerged [1]. Bitcoin, created by Satoshi Nakamoto, was only possible thanks to the development of the Blockchain technology, which was introduced along with Bitcoin in the same paper [2]. This technology have become specially relevant in recent years for two reasons: first because it is the base of most of the cryptocurrencies, and second because it is used for other important systems such as Smart contracts [3].

2.1.1 Blockchain

Blockchain is just a distributed database of records, or in other words, a public ledger of all transactions that have taken place among participating parties. Each time that a transaction is executed, it has to be verified by consensus of a majority of the participants. If it is correctly verified, it is entered into the system and can never be removed, since Blockchain keeps the information of every transaction ever performed [4].

The intrinsic structure of Blockchain defines its characteristics. In Figure 1 the basic structure of Blockchain appears, consisting of a sequence of blocks where each block maintains:

- *Block Header*: Metadata of the block such as a timestamp and the hashing number of the block, *i.e.* the own identifier, which is the hash value of all the transactions located in the block.

- *TX*: transactions data.

- *Parent Block Hash*: the hashing number of the previous block, *i.e.* the identifier of the previous block of the sequence.
- *Transaction Counter*: a transaction counter.

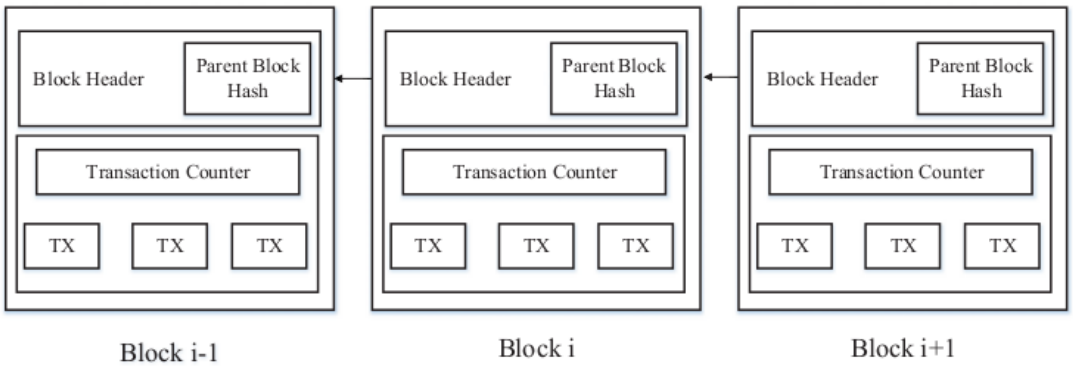


Figure 1: Structure of Blockchain

Source: [5]

The number of transactions that can be stored on each block depends on its size and the size of each transaction [5]. Thus, new blocks have to be built from time to time in a process known as mining.

In the mining process, the system asks some users, known as miners, to solve a very complex mathematical problem, also know as Proof-of-Work (POW), needed to complete and group all the new transactions in a new block, which is then linked to the sequence. The first miner that finds the correct solution is rewarded, typically with an amount of the mined cryptocurrency [6].

2.1.2 Illicit crypto mining

In Figure 2 we can observe how three of the most commonly used cryptocurrencies have acquired a significant price in recent years, even though their value have decreased lately compared with their peaks. The high value of cryptocurrencies, added to the previously mentioned fact that only the first user who mines a block is rewarded, have

transformed the mining process into a tough competition.

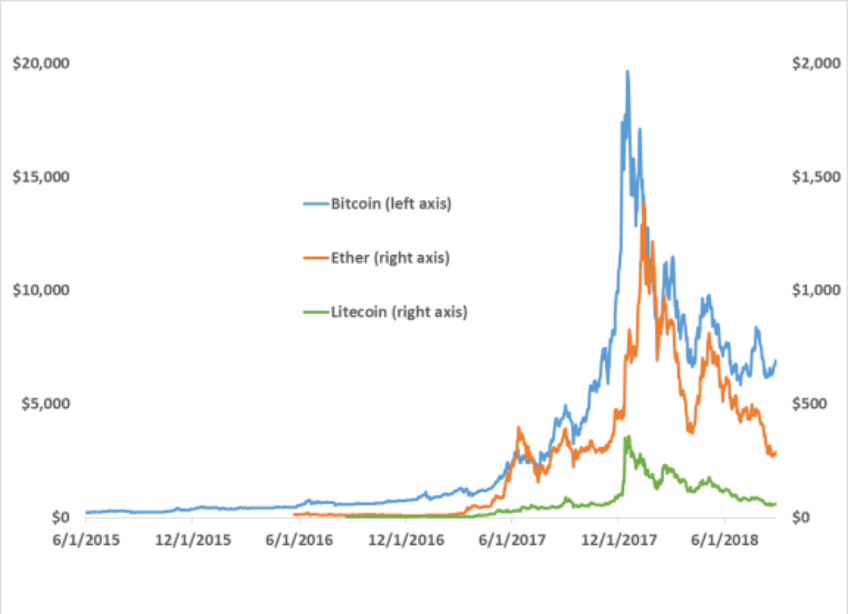


Figure 2: Evolution of cryptocurrency values

Source: Coinbase

Since Mining requires a noticeable number of computing resources, some miners decide to collaborate and work together in what is called pools of miners, with the aim of reaching higher levels of performance, splitting then the reward, if obtained, between all the participating miners. A part from increasing their opportunities using this method, some malicious miners may decide to hijack resources from other users in order to gather more computing power, so as to have an even bigger chance to be the first ones to successfully mine a block without having to divide the reward [7]. This action, which is known as illicit cryptomining or cryptojacking, is typically conducted using one of these two methods:

- **Browser-based cryptomining programs**: the mining process is executed in scripts embedded in web content.
- **Binary-based cryptomining malware**: the mining process is embedded in the payload of a malware running on an infected machine.

The same attack pattern is perpetrated in both methods. The attacker infects hundred of machines and use all of them to obtain a hash-rate, *i.e.* the speed of the miner's performance, that can reach similar levels than medium-sized mining farms [7].

Each method has different characteristics. In browser-based cryptojacking, the damage is easily avoidable, since the victim only needs to stop browsing the malicious site, or otherwise restrict the use of *JavasScript*. Whereas cryptomining malware is harder to detect and remove, since it involves classical malware features, like persistence and obfuscation [7].

The problem arises when it comes to detect these cryptojacking activities, specially the binary-based one (since it is normally obfuscated), and this is exactly the main purpose of this project, the detection of cryptomining malware activity. The aim is to build a system which, using different network indicators and measurements obtained from reports of other applications, detects cryptomining activity. This system will consist of three different machine learning systems based on three different features of cryptomining malware.

2.2 Stakeholders

As it was mentioned above, cryptocurrency mining requires considerable resources, what means that a cryptomining malware will decrease significantly the performance of infected machines, since the majority of resources are deflected to run malicious mining processes without the victim's knowledge. For this reason, every end-user would be interested in the system proposed in this project, since this huge use of resources implies a significant energy consumption, which, consequently, decreases the life expectancy of these machines.

Therefore, the solution that seeks this project is particularly interesting for companies and, in particular, for the network administrators of these companies. In every company is very important that machines reach an optimal performance, so a malware decreasing it means less working capacity or, in other words, less production. This

is specially significant for ICT companies, which base their economic activity on their machines. In addition, a malware consuming a noticeable amount of energy is directly translated into a considerable economic loss for the company. But the reasons do not stop there, because if an administrator is even able to detect the infected hosts, it would be easier to find the cause as well as the most appropriate solution while other works can keep working.

Additionally, this project may be significantly helpful for the field of cybersecurity. The number of cryptomining malware have become one of the biggest threads in nowadays cybersecurity, so a reliable system that identifies this type of threads, as it is expected from this project to be, would suppose a substantial improvement in cybesecurity against these threads, offering a huge help to decrease and stop the damage that they may produce [8].

Finally, this project will be truly useful for me as a student and future professional. Not only because it gives me an excellent opportunity to study and work with a contemporary problem, but because it offers me a unique way of approaching to an important topic.

2.3 State of the art and justification of the selected alternative

There is not a standard method to perform the detection of cryptomining on networks. However, the detection of cryptomining activity is an important research topic within networking and cybersecurity, hence many approaches can be found currently, all of them with their benefits and drawbacks.

Some of the existing approaches analyse the code of web applications in order to find instructions frequently present in mining algorithms [9]. Another approach consists in analysing the memory consumption in web applications to compare it to the memory used by known mining processes [10]. Both of these previous solutions require a big effort due to their nature, so they are quite expensive in terms of resources, since they demand a constant analysis of the instructions executed or the memory consumed.

Additionally, there are many other studies that suggest the analysis of flows in order to detect traffic anomalies that might indicate the presence of miners [11]. This approach cares about performance, therefore it becomes a very feasible method, since it does not need a huge amount of resources and, consequently, does not need to upgrade or change the structure of the network. Nevertheless, using just the flow may not be enough, since the recognition of mining activity through flow is possible when the *Stratum* protocol is used. This protocol is typically the one that is utilised when miners want to connect to the pool, and due to its features, it is easy to recognise. However, there are other mining protocols that may be used instead of *Stratum*, such as *getblocktemplate* [12], so in these cases this method is not enough.

Currently there are many programs that try to detect cryptomining malware. For instance, the market is full of anti virus, such as *Avast* and *Malwarebytes* [13] [14], that will detect existing cryptomining malware located on the machine with high reliability [15]. There also exist some anti browser-based malware extensions for browsers, such as *No Coin* and *Anti Miner* [16] [17]. Nevertheless, there are several drawbacks on all of these already existing solutions:

- The user of the infected machine has to be aware that the machine might be compromised, and in order to do that, normally an exhaustive analysis of the machine is required, meaning that resources will be taken for some time.
- The user of the infected machine has to install these programs or plugins and, normally, they are not free, which is translated into an outlay that may not be negligible, even more if we consider the case of a company, where each machine have to install those programs.

Finally there are two more traditional approaches, which are the Deep Packet Inspection (DPI) and the IP address filtering. When a DPI method is required, an exhaustive inspection of each packet has to be performed, which means using a huge amount of resources, what makes this solution an unfeasible one for medium/large networks. In the second method, the network has to be altered so as to add a filter that avoids all the

connections to IPs of known pools. Furthermore, miners connecting to unknown pools will not be recognized, so this solution is not very convenient.

For all these previous reasons, it is sufficiently justified the idea behind this project, a system that detects the activity of cryptomining malware without the need of individual analysis of each machine and several program licenses in case of companies. The only requirement would be to gather information from the network in terms of flow and DNS queries, something that an administrator could easily do.

3 Scope of the project

3.1 Objectives of the project

Two main objectives motivate this project. The first one is to understand how cryptomining malware works and how it concerns network's traffic. The accomplishment of this goal will be achieved by fulfilling a group of sub-objectives:

1. Understand how malware usually works.
2. Recognise the types of cryptomining malware.
3. Comprehend the characteristics of each type of cryptocurrency mining malware.
4. Know how network's traffic usually is.
5. Collect several network measurements and traffic traces from infected networks.
6. Infer how each cryptomining malware alters network's traffic.

There are two main obstacles that may be found during the execution of this first objective. First of all, it may be hard to quickly and completely understand the features of each type of cryptomining malware, since malware is constantly changing and evolving, which means that more time will be required to research and learn about it. A part from that, it may be the case that the cryptomining malware alterations from the captured network's traffic are not easily distinguishable. If this happens, more measurements will be collected until one of these two conclusions is reached: finally it is possible to detect it with the new data, or it is not possible to detect it, so a more precise inspection of the traffic is needed.

Once that the first objective, with its corresponding sub-objectives, has been completed, the second one is considered afterwards. This second objective is to develop all of the three machine learning algorithms. The first machine learning algorithm, named

Pool Method, receives a list of the URLs that have appeared on DNS resolutions, and it indicates if each URL is a pool or not. The second method, also known as *Malicious URL Method*, receives a list of all URLs that were part of some DNS resolutions, and it identify which of them might may be malicious. Finally, the last method called *Flow Method*, receives network measurements and identify whether there are flows indicating mining activity. As before, the correct execution of this second objective will require the achievement of some sub-objectives:

1. Combine the data collected in the first objective with already existing data in order to create a good dataset for each method.

2. Build different models for each method and train all of them.

3. Test the accuracy of the considered models and select the best one for each method.

Only one potential obstacle may be found during the fulfillment of this objective, which is that perhaps the collected data is not enough to correctly train the models. This can be solved by collecting more data.

3.2 Potential obstacles and risks

During the development of this project, some obstacles are susceptible to arise. In general terms, these are the potential ones:

- **Timing:** The time to work on the project is delimited, which means that only a short amount of time can be dedicated to each task.

- **Lack of awareness:** Some concepts of this project, and therefore some technologies, are completely new to me. This translates into several hours of research and lecture, what would possibly slow down my performance.

- **Bad solutions:** It is possible to find that some of the methods are not good enough and they are unable to perform their task accurately. If that is the case, either a

new approach will be suggested for the corresponding method or, if that is not possible, the method will be discarded.

- **Collection of data:** An important step during the development of my project is the collection of data for analysis and training purposes. This data may be hard to gather.

3.3 Methodology and rigour

The development of this project will follow the traditional waterfall procedure. Specifically, the procedure has been planned in different stages:

1. Study previous cryptomining detection related works. This stage is specially important, because it will be necessary to understand how mining processes can be identified from network measurements or traffic, as well as what tools are useful and efficient for this purpose.

2. Study and understand how cryptomining malware works, focusing on what are the similarities between them and which unique features they have.

3. Gather data, in form of traffic traces and network measurements, from networks where cryptomining malwares are working, as well as clean networks, *i.e.* networks without cryptocurrency mining. Networks with both traffic of non malicious applications and cryptomining processes are also considered.

4. Filter and analyse the data in order to extract conclusions about them. Basically the intention behind this is to recognise patterns and common characteristics that will help to determine when some measurements indicate that the network is being used to mine cryptocurrencies.

5. From all the previous data and conclusions, prepare a dataset and combine it with already existing datasets. Implement and train the three machine learning algorithms mentioned previously in this document.

6. Test the reliability of all the models previously built, and select and redefine the best ones if needed.

7. Finally, extract conclusions and prepare the defense of the project.

3.3.1 Developing tools

As it has been mentioned above, before implementing the machine learning algorithms it is needed to gather data from network. In order to do that we will rely first on *VirtualBox* to create an infected host working on the network. This virtual machine will be based on *Windows 7*. With the environment set, the collection of data will be done using *NetFlow* protocol, which performs flow-level measurements. To collect these measurements, *nfcapd*, *softflowd* and *tcpdump* will be used. Additionally, to capture traffic traces from where DNS queries will be extracted, *tcpdump* as well as *WireShark* will be needed.

Once that all a significant amount of measurements have been collected, the next tool that will be used is *Python* in order to sanitise and analyse the data. Additionally, the same language will be employed to implement the machine learning algorithms. The reason of choosing *Python* was the fact that this language offers several modules oriented to Machine Learning, which will facilitate the entire development process. Another important reason for choosing *Python* is that it is widely used and accepted, which means that many communities and forums will be available to consult. The operating systems that will be utilised during the project are *Ubuntu* and *Debian 9*.

In addition to the above stated tools, *Team Gantt* will be utilised to generate the Gantt chart, and *Github* will be used to track the code and verify it each time that it changes. Moreover, *LibreOffice Impress* will also be employed so as to prepare the slides to defend the final project.

3.3.2 Monitoring tools

In order to ensure that all objectives are met, some monitoring processes will be applied:

- Firstly, so as to ensure that the data has been correctly gathered, it will be sanitised first before their use.
- Secondly, data will be analysed to check if the studied concepts about cryptomining malware correspond to the reality.
- Thirdly, to correctly categorise URLs as Malicious (for dataset corresponding to *Malicious URL Method*), an URL will be considered as malicious if at least 10 engines detect it as malicious in *VirusTotal* [18], or if there is at least one existing report (hybrid or static analysis) indicating that it is malicious. The source for this reports will be *Hybrid Analysis* [19] for hybrid analysis and *Infosec* [20] for static analysis.
- Fourthly, to achieve a satisfactory level of prediction, the machine learning algorithms will be trained using a large enough data. In addition, hyper-parameters of machine learning models will be selected using an exhaustive grid search. This will ensure that the best value for each hyper-parameter is selected.
- Finally, the machine learning algorithms will be tested to check if they reach a good level of accuracy.

4 Time planning

The project is expected to be elaborated in a four-moth period. It officially starts with the beginning of GEP course and it approximately ends 4 months after that.

The academic load of this project corresponds to 18 ECTS credits. According to the official regulations of the End of Degree project in FIB [21], each credit is estimated to be 30 hours of work, which means that the total length of the project is equivalent to 540 hours.

This section focuses on the planning, organization and management of these previously mentioned hours. In order to do that, these working hours are distributed among all the required tasks. This schedule is expected to be followed, since it tries to be accurate and realistic. However, it may suffer changes during the progression of the project if it is necessary.

4.1 Description of tasks

The first step is to identify and describe all the tasks required to develop the project, therefore this is what this section does. The resources needed for their fulfillment, the estimated dedication in hours and the order of their execution are also explained here.

4.1.1 Project management tasks

Here are gathered all the tasks dedicated to the management. These tasks are crucial, since they define and build the structure of the project.

Here we can find the tasks needed to be accomplished before any other else, *i.e.* definition of the project's scope, planning, budget and sustainability report.

In addition to the previous tasks, here are also located two more that have to be

continuously performed. They are the documentation of the project, and the control and monitoring meetings. These tasks are expected to be executed in multiple occasions during the development of the project.

The resources that will be used in order to complete these tasks are the software *Trello*, which will be very helpful to manage and organise the execution of tasks, the software *Team Gantt* to generate the Gantt chart for time planning, and *LaTex*, which will be used to document the entire project. Regarding the human resources, a project manager will be needed to work on these tasks.

4.1.2 Study of cryptomining malware

This project is mainly a research one, which means that a significant amount of hours have to be dedicated to study the fundamentals of cryptomining, as well as the current existing malware to perform cryptojacking.

No special resources are needed here, seeing that this task only consists in research. The only resource necessary is someone who does that research, but this could perfectly be the project manager.

4.1.3 Preparation of the environment

Once that the main cryptomining malware methods and tools have been completely understood, the next step consists in gathering network traffic from an infected network. In order to do that, first a controlled infected host has to be prepared under a network.

The infected host will be built using virtual machines with the support of *VirtualBox* software, and infected with different samples of crytpomining malware. To ensure the security, the free VPN *windscribe* will be used. Human resources are required now so as to create the desired environment.

4.1.4 Collection of data

With the environment correctly prepared, the network's traffic and flow will be collected. The aim is to analyse them in order to detect patterns and features that would help in the subsequent recognition of this malware.

Traffic and flow will be collected several times in many different conditions, since they will be helpful to have a big enough dataset. Human resources are needed in order to collect this data using *tcpdump* and the *NetFlow* protocol. To do it, *softflowd* will be used as exporter, *nfcapd* will be used as collector, and *tcpdump* will be used to read and extract data from the collected file.

4.1.5 Analysis of data

As mentioned above, this task focuses on analysing the captured traffic and flow with the final intention of detecting patterns and features. This step also includes the sanitation of data.

With this information and the studied one, the objective is to understand how malware can be detected, or in other words, where is the content that indicates the presence of malware.

The resources required here are mainly three, someone who is able to analyse the data, and *Python* and *WireShark*, which will be used to do it.

4.1.6 Build the machine learning methods

At this point, the penultimate step is just the development of three machine learning algorithms able to identify if the presence of a cryptomining malware.

This step includes the creation of the required dataset, the selection of the most suitable model, the training of it, and the verification of the good reliability level of the

system.

An engineer will be required here to develop the system. Additionally, the resources that this person will use are *GitHub*, in order to track the code, and *Python*, which is the programming language that the engineer will utilise to build the system.

4.1.7 Results and defense

This is the last task. This one is only dedicated to extract conclusions from all the previous work and prepare the defense of the project.

The only tool required here is *LibreOffice Impress*, which will be used by the project manager in order to prepare the slides for the defense.

4.2 Resources

Here is presented a list of all the resources that will be used during the development of this project.

Regarding the hardware, the project will be elaborated using mainly an Asus PC with the next relevant features: 16 GB RAM, GPU nVidia GeForce GTX 1050 4 GB Video RAM, and a CPU Intel Core i7-7700HQ 2.80 GHz.

Regarding the software, many tools will be used for different purposes, as it has been stated in the previous section. In order to keep an updated list of uncompleted tasks and tasks in progression, *Trello* will be employed. For documentation, the software used will be *LaTex*, and *Team Gantt* just for the elaboration of the Gantt chart. For the collection of data, *VirtualBox* with *Windows 7* is the choosen software for the preparation of the environment, which will be installed on *Debian 9* and and will be run while using the VPN *windscribe*, and *tcpdump* along with *NetFlow* are the tools that will be utilised to collect both traffic and flow, using for this last one *softflowd* as exporter and *nfcapd* as collector. In addition, *Python* will be the language used to define the machine

learning program as well as to analyse the collected data, that it will be also done using *WireShark*. Finally, *LibreOffice Impress* will be the tool to prepare the project defense. The development and execution of code as well as the analysis of data will be run using the *Ubuntu* operating system.

4.3 Time planning

In Table 1 each task appears with its corresponding dedication time:

Task	Estimated Time in hours
Project management tasks	150
Study of cryptomining malware	70
Preparation of the environment	80
Collection of data	25
Analysis of data	40
Build the machine learning system	130
Results and defense	45
Total	**540**

Table 1: Estimated time per task

4.4 Sequence of development: Gantt

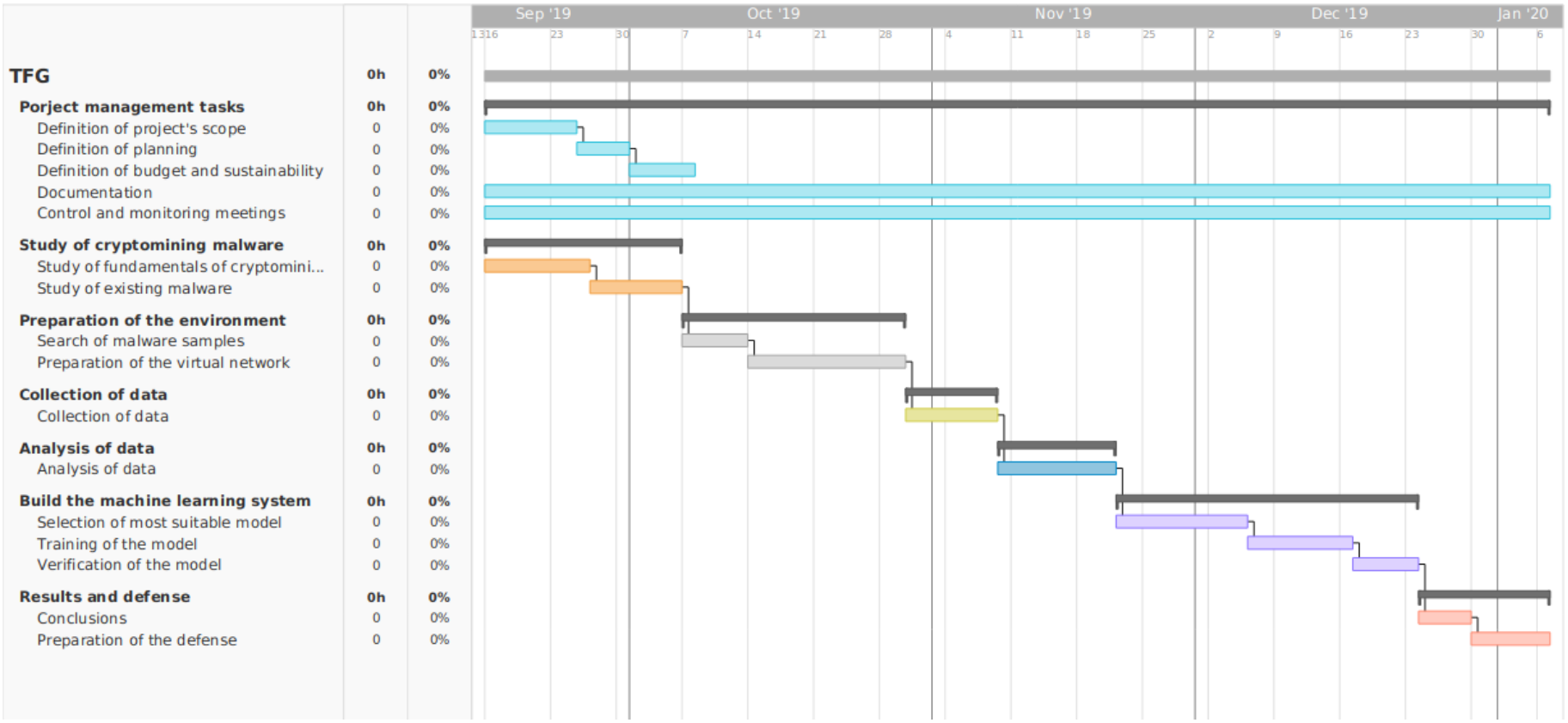


Figure 3: Gantt chart

4.5 Action plan and obstacles

As it can be observed in Figure 3, the intention is to keep documenting the project beside the development itself of it. The same works for control and monitoring meetings, which are expected to be taken every one or two weeks during all the project development. Weekend days do not appear in order to simplify the chart, but they are considered for the computing of total hours.

One of the potential obstacles previously noticed was the lack of time to complete a task. If a task requires more hours than expected, next task dedication will be reduced in the same way, so as to avoid lack of time to finish the project. Nevertheless, if a task is completed before the expected ending time, the next one will start immediately.

The lack of awareness was another potential problem, since many hours will be dedicated to the research. In case of needing more time to study some of the essential topics for the project, more hours will be dedicated to the project, increasing the total workload.

Regarding data, a few obstacles may be reached. The first one is that the collected data is not enough to extract conclusions about the alterations of cryptocurrency mining malware on the network's traffic. If this happens, more data will be collected until it is possible to do it, or in case it is not, a more precise inspection of traffic will be used. This is translated into inspecting each packet more accurately. The other obstacle may be the need of more data in order to train the machine learning methods. In this case the solution only consists in the collection of more data.

Finally, in a previous section it was mentioned that it is possible to not reach a good accuracy for some of the models. If this happens, either an alternative approach will be studied, or if this is not possible, the model will be discarded if the other ones are still functional with good accuracy.

5 Budget

Once that the objectives and the development planning have been clearly defined, a new decisive stage immediately emerges, which is the analysis of the budget. The aim of this section is to study this crucial aspect of project management, obviously applied to this specific project.

In this section the costs are firstly identified, and secondly estimated.

5.1 Hardware

Here are considered all the costs related to the hardware used during the development of the project. As it was previously mentioned, only one PC will be utilised, and its characteristics were already specified. Additionally, one peripheral, an external mouse, will be needed, but since it is not expensive and it will be used exclusively with the PC, its price will be included in the total cost of the PC.

Table 2 contains the cost of each hardware element as well as the total cost of hardware, which is the same in the case of this project. The amortization is expressed monthly, since it is computed considering that the TFG is developed in a four-moth period.

Component	Cost	Lifetime	Amortization
PC	1,050 €	8 years	43.75 €
Total	**1,050 €**		**43.75 €**

Table 2: Hardware costs

5.2 Software

Working with different software may mean a serious outlay. Owing to this fact, it is helpful to analyse their cost, which is exactly what Table 3 does by enumerating all of them and specifying their cost, in conjunction with computing the total cost of software. However, as it can be noticed, all the software managed in this project is free, which is translated into a null software outlay.

As a clarification, the *Windows 7* that will be used with *Virtual Box* is an already prepared machine distributed for free by *Microsoft* [22].

Software	Cost	Lifetime	Amortization
Ubuntu 18	0 €	4 years	0 €
Debian 9	0 €	4 years	0 €
Python	0 €	4 years	0 €
Trello	0 €	4 years	0 €
LaTex	0 €	4 years	0 €
Team Gantt	0 €	4 years	0 €
LibreOffice Impress	0 €	4 years	0 €
VirtualBox	0 €	4 years	0 €
Windows 7 machine	0 €	4 years	0 €
windscribe VPN	0 €	4 years	0 €
NetFlow	0 €	4 years	0 €
nfcapd	0 €	4 years	0 €
softflowd	0 €	4 years	0 €
WireShark	0 €	4 years	0 €
tcpdump	0 €	4 years	0 €
Total	**0 €**		**0 €**

Table 3: Software costs

5.3 Human resources

First of all, the human resources required are identified in Table 4, indicating the cost of each one along with the total cost of human resources. Afterwards, the work of each of them is detailed and situated in some of the tasks mentioned previously.

Positions	Price per hour	Total working time	Total cost
Project Manager	68 €	195	13,260 €
Systems administrator	14 €	150	2,100 €
Data scientist	18 €	65	1,170 €
Machine learning engineer	18 €	130	2,340 €
Total		**540**	**18,870 €**

Table 4: Human resources costs

Project manager is probably the most important position in this project. The project manager, as it can be observed, assumes the biggest workload, working on the project management tasks and the definition of results and preparation of the defense. Additionally, the price per hour of this position is quite high, according to the average salary in Spain [23]. For all the previous reasons, a majority portion of the budget for human resources is destined to pay this position.

A Systems administrator is required in order to study the cryptomining malware so as to prepare the environment where the network measurements will be collected. In other words, the systems administrator will work on the two tasks oriented to prepare the environment for the data collection. This is translated into 150 hours of work, which will be remunerated with 2,100 €, 14 € per hour, the average salary per hour in Spain for this position [24].

Finally, two more roles are needed, a data scientist and a machine learning engineer. Their responsibilities are very well defined, the first one has to collect the data and analyse it, while the second one has to prepare a machine learning software able to

perform the expected classification with a good reliability. Their corresponding salaries are 18 € for both of them, the average national salary in both cases [25][26].

To summarise the above mentioned work dedication per position, in Table 5 each role is attached to the task/s that each role will need to work on, indicating the number of hours that will be required, *i.e.* the estimated duration of the task. In addition, the total estimated cost per task is shown. The tasks are the previously identified on Gantt chart.

	Dedication time in hours				Cost
	Project Manager	Systems administrator	Data scientist	Machine learning engineer	
Project management tasks	150				10,200 €
Study of cryptomining malware		70			980 €
Preparation of the environment		80			1,120 €
Collection of data			25		450 €
Analysis of data			40		720 €
Build the machine learning system				130	2,340 €
Results and defense	45				3,060 €

Table 5: Positions attached to their tasks and total cost per task

5.4 Total budget

In the previous subsections the main costs that have to be considered in every computer science project were analysed. Other costs considered negligible, such as transport, have not been included.

Here all these previous costs are added in order to obtain a global and simplified vision of the total budget required. Table 6 expresses this concept.

5.5 Incidents, contingencies and management control

During the development of this project there exists the possibility of facing some obstacles, which means that the original budget may suffer variations.

Resource	Cost
Hardware	43.75 €
Software	0 €
Humans	18,870 €
Total	**18,913.75 €**

Table 6: Total costs

It was mentioned that a possible problem is the modification of the expected time dedicated to work on some of the tasks. If this occurs, the budget destined to human resources will vary according to the corresponding time modification, *i.e.* if more time is required to complete a task its corresponding worker will receive an extra remuneration proportional to the number of extra hours dedicated, and the same works the other way around.

Regarding the hardware costs, the unique possible problem is the damage to the PC used for the development of the project. In that case, the budget for hardware will increase in order to pay the repair or the acquisition of a new machine. The worst situation would be the need to acquire new hardware, which will translate into an approximate increment of 1,050 €, the price of the currently used hardware.

Software costs are hardly sensitive to increase, since the software needed is free and widely used, which means that the necessity to find an alternative is highly improbable, but even in that case, several free alternatives exist anyway.

Consequently it seems that the original budget will not suffer huge modifications, so the final budget will remain close to the original one.

6 Sustainability and social commitment

Sustainability in project management is also essential, specially for technological projects where the impact on the ecosystem may be significant. In addition, a smart use of our finite resources is certainly needed in order to not waste them. For these reasons, this section tries to clarify these topics in the context of this project.

In order to analyse the sustainability of this project, it will be detailed on three different areas: economical, social and environmental. Furthermore, a conclusion of the self evaluation survey is also included.

6.1 Social sustainability

6.1.1 Project put into production (PPP)

First of all, I feel like this project will be extremely beneficial for my personal growth. First of all, because it will give me an excellent opportunity to experience and learn how to handle a medium/big project, providing me tools that I may use in a future. Moreover, this project will offer me the possibility of approaching to a very appealing topic to me.

Additionally, the opportunity to work on a project within the field of cybersecurity have brought some reflections to my mind. These reflections are related to the ethics behind the exploitation and subsequent damage of the property of other people for profit. However, the ones that are making profit from other users' damage are not only the attackers, but also the distributors of anti malware software, since the majority of them are selling their products at a very high price. For this reason, I think that users should have the possibility to protect themselves against threads like the ones studied in this project.

6.1.2 Lifetime

This project has a significant social impact, since it offers a solution to an existing problematic that affects thousands of PC users all over the world. Actually this is what justifies this project, the real need of a solution in order to stop an illicit activity that damages a noticeable number of machines.

Currently this above mentioned problem is solved using anti malware software, which is normally quite expensive, that has to be installed on every computer of the network. This is translated into a serious outlay, not only for companies but also for individuals. Here appears the social impact that it was introduced earlier in this section, this project offers a solution that is cheap, that only requires to be installed on one machine of the network, and that it will help to avoid serious damages on other computers of the network. This is the improvement of social dimension of this project with respect the other existing solutions, and at the same time, another justification for its need.

As it has been stated before, the main beneficiaries of this project are both the regular computer users and the companies, specifically the network administrators, who are the ones in charge of security and the proper working of the networks inside the companies.

On the contrary, the main affected parties are basically the attackers using these kind of attacks, since this project offers a countermeasure, and anti malware distributors, since this project can be seen as an alternative product. Regarding the first group, they will probably need to upgrade their techniques to ensure avoiding the detection. About the second collective, it is true that this project presents a cheap an reliable alternative to expensive anti malware, however, the anti malware distributors will not be really affected, due to the fact that this project only focuses on one specific type of thread, whereas anti malware software normally offer protection against many others, so a majority of users interested in full protection will still purchase their products.

6.1.3 Risks

This project will be mainly detrimental for cybercriminals. Nevertheless, there exists a risky scenario where this project can be harmful for the main beneficiaries of it. This scenario happens when the attacks change and new techniques are applied (for example using non common protocols or when the use of proxies or private pools is generalised). In that case, the solution proposed in this project would start failing and consequently, would stop offering an accurate detection. Consequently, administrators and users trusting this solution to detect attackers will be vulnerable.

There is no other possible risk associated to this project, since there are not more dangerous scenarios. Additionally, the dependency of users to this solution is not excessive, since there are still other alternatives.

6.1.4 Conclusion

The evaluation of this project in this field is an **8**, because it has a considerable impact on society and it really improves the existing solutions so as to avoid a serious problem. Furthermore, the risks are few and there are not many affected groups in a negative sense.

6.2 Economical sustainability

6.2.1 Project put into production (PPP)

In previous sections the total estimated budget was examined, splitting it into budget for human resources, for software and for hardware. This estimation tries to be the most accurate possible, even considering potential deviations and contingencies.

The cost foreseen tries to be the lowest possible, since the only cost considered is the one related to hardware and human resources, and both are completely necessary. In

order to achieve that, only free software was considered, since it can be quite expensive otherwise. For instance, just deciding to use *Linux* instead of *Windows* there is a saving of 145 € (price of the cheapest version of *Windows 10* [27]). In addition, the cost related to human resources is also the smallest one, since each worker only works the strictly needed amount of time.

This system would be very helpful and necessary for companies, universities and, in general, for every network accessed by multiple users. This added to the benefits that it offers to users with respect to the other existing solution, will permit to establish a good selling price to cover all the costs and even earn profit. This system will be sold to companies or individuals using an annual license.

6.2.2 Lifetime

The solution introduced in this project will economically improve the other existing solution in the sense that it only focuses on this type of malware, meaning that there will not be the need to spend as money as the other solution spends in studying and analysing all new malware. This will also give the opportunity to sell the system for a lower price than the other solutions, since less costs have to be covered. This is specially important during the lifetime, since the upgrade will be less expensive.

Nevertheless, there will be some cost related to the maintenance of this solution, since the threads are constantly evolving and cryptomining attacks constantly evolve. For that reason, human resources will be required to keep updating the detector, which means a constant outlay. In particular, a project manager and an engineer are the needed profiles, which suppose, as it was seen before, a cost of 18 € per hour to pay the machine learning engineer, along with the 68 € per hour normally paid to the project manager.

Another important cost that may be considered is the renovation of the hardware, which has to be replaced at the end of its lifetime or in case that it breaks. However, if it is possible to repair it and it is cheaper than updating it, this option has to be contem-

plated.

6.2.3 Risks

Despite the good economical viability of this project, there is an important risky case that might compromise the project. This hypothetical situation would happen if the value of cryptocurrencies drops to the level that makes mining no longer profitable. Since attacker needs some investment in order to produce and distribute malware, if this process stops being profitable, no more resources will be assigned for this purpose, so this kind of attacks will stop. If that is the case, a detector will become useless.

6.2.4 Conclusion

This project is evaluated with a **6** in this field. Costs are minimum and necessary. However, there are some alternatives offering protection and/or detection to more threads than this project. In addition, there also exists an important scenario that might ruin the entire project, despite being very unlikely to happen.

6.3 Environmental sustainability

6.3.1 Project put into production

This project does not require any special resource, since it is mainly a research one with an implementation of three machine learning algorithms. This means that regarding environmental sustainability, the only resources used are the electricity consumed by the PC, which can be estimated at 230W [28], and the paper required to print some information.

In order to avoid the excessive electricity consumption, which production releases huge emissions of CO_2, a possible solution would be the reduction of PC usage, mean-

ing that the expected time dedicated to the study of cryptomining malware would be reduced in a significant way. The major drawback in that case would be a lack of knowledge. Another way to reduce the use of resources, which has been applied, is the substitution of printed paper for electronic documents.

6.3.2 Lifetime

The resources that will be needed during the lifetime of the project is just a PC, since nothing else is expected to be used. This means that the environmental impact regarding the updating and maintenance of the project is just the production of the electricity required for this PC. Since the PC utilised will be the same than the one used during the project put into production, the consumption will remain approximately around 230W.

This project indirectly allows the reduction of some resources. Specifically, the electricity consumption might be reduced, thanks to the fact that if cryptomining malware is detected thanks to this project, the power consumption of the victim's machine will be reduced once the miner is removed, so the CPU usage will be restored again to normal values. For this reason, the project can be considered as positive for the reduction of the environmental foot print, since the resources are few and the probability of reducing the power consumption of some machines quite high.

6.3.3 Risks

There is a scenario where the environmental foot print might increase. This is the case where the project is working but not detecting any miner. In this case, the works of updating and maintenance of the project would be still ongoing, *i.e.* consuming electricity, but not at the same time the project would not be helping to reduce the usage of any other resource as mentioned before.

6.3.4 Conclusion

This project receives an **8** in this area, since there are only a few necessary resources, of which only one was finally required (electricity).

6.4 Matrix of sustainability

The scores assigned to each dimension of sustainability are summarised in Table 7.

	Social	Economical	Environmental
Score	**8**	**6**	**8**

Table 7: Matrix with the corresponding score for each dimension

With the previous analysis, it seems a very balanced project in terms of sustainability. This is due to the small number of resources required to develop the project as well as the significant impact that it has, not only on society by helping users and companies to protect themselves against miners, but also on the environment by helping with the reduction of power consumption.

6.5 Conclusions of the self evaluation

After completing the survey about my domain of sustainability, I have realised that my general knowledge of managing a project caring about sustainability is quite weak. I have realised that I have no problem solving IT related problems and managing a big project, defining its budget and, in general, its features, but when it comes to analyse the impact of that project in the economy, the environment and the society, I find myself with troubles to completely understand it.

I think that this is due to the fact that during my degree, the courses were always focused on the development of projects, just evaluating if each project was correctly suggested and implemented. There was never a real need of analysing the sustainability

behind those projects. Sustainability was only emphasised on some of the "Competències Transversals", where I feel that I learned a majority part of what I know now about that topic.

A part from learning about sustainability in the previous mentioned way, the development of this document has also pushed me to study more about this topic. In other words, I can say that what I know now about the topic only comes from this research for this third document and these "Competències Transversals".

For this reason, I feel like I still have a lot to learn. For example I know nothing about the deontological ethics that are mentioned in the survey, so I do not know how to boost an IT project with coherence in an economical, social and environmental sense. Consequently, I would suggest to include more actively this topic on FIB courses, in order to correctly form the students so they can work considering the sustainability.

7 Data overview

7.1 Justification of the required data

As it was previously mentioned, the idea is to develop three different machine learning algorithms, each one of them will focus on a specific feature of cryptomining malware. These features are: connections to mining pools, connections to malicious URLs, and network flow.

The connections to mining pools is a very specific and helpful feature for the detection of miners, since every mining process has to connect to some pool. However, this is not enough, as some miners may connect to private unknown pools or through proxies. In fact, this last technique is quite common in modern attacks, due to the fact that some pools block users when they are using botnets, *i.e.* a net of infected machines mining for one user, so instead of generating hundreds of hash-rates for one user, with proxies the attacker can send just one hash-rate to the pool, avoiding the detection of botnet usage [7].

Another important characteristic is the connection to malicious URLs. This is interesting because some cryptomining malware is distributed using other malware, which is called *Droppers*. *Droppers* connect to malicious URLs in order to download other malware such as cryptomining malware or spyware [29]. For this reason, it is interesting to identify these cases to detect miners. Nevertheless, in some cases it is not easy to detect whether an URL is malicious or not without inspecting its content, so even with a machine learning method with high accuracy, some URLs may be ignored.

Finally, the network flow is the last important characteristic that we want to consider. A majority of the pools use *Stratum* protocol in order to communicate with miners. This protocol, which is implemented over TCP, consists of two sets of methods, one from client to server, and the other from server to client, and all of these methods use JSON [30]. Additionally, *Stratum* connections are characterised by the fact that the

server usually transmits data constantly whereas the client transmits few data, which makes this type of connections easy to recognise. Despite this protocol is the most commonly used nowadays, there are some other possible miner-pools protocols which differ in a significant way, and this is the major drawback of this method of detection, it only works for *Stratum* or other JSON related protocols with similar patterns.

After noticing all the above stated drawbacks and benefits, the conclusion is that the best option is to gather and apply all of the three methods on the same system. This will increment the probabilities to detect miners even in case that one or two methods fail. For this reason, we need the data required for all the three methods, in order to analyse it, so as to guarantee that it is possible to detect patterns and behaviours that will help to identify miners, and to prepare the required datasets.

7.2 Required data

In the previous section, it was identified that the data required consists of URLs and network flows. Regarding the first group, URLs, the best alternative is to check the DNS queries, since this is the most efficient way to check which connections are tried by each host. An alternative to identify the URLs would be to collect all the traffic and identify and inspect the HTTP messages, but this will require a deep inspection of each packet, which is translated into a more expensive solution. In contrast, in order to check the DNS queries, the only file required is a *dnstop* file containing all the DNS resolutions. This is something an administrator can easily obtain.

Regarding the network flow, Jordi Zayuela's paper [6] will be really helpful for this task, since it will be used as a reference for this method. *Netflow version 5* would be the chosen solution to gather it. *Netflow* is a protocol developed by *CISCO* that analyses the network traffic. This protocol joins all packets with common features. In other words, all packets sharing the same source and destination IP address, the same source and destination port, the same protocol, the same ingress interface and the same IP type of service are gathered into an unidirectional single flow [31].

The version of *Netflow* used in this project allows to export several fields of information, however, due to the fact that the main feature that would possibly allow the detection of *Stratum* protocol is related to the quantity of information transmitted, we are only interested on those fields containing information about quantity of data, as well as host address which will be useful for the identification of the mining host. Additionally, since *Stratum* does not use any specific port by default, information about ports is not considered. For these reasons, the only information that we are exporting is the following:

- Source IP
- Destination IP
- Duration
- Number of bytes
- Number of packets

As it was stated before, the quantity of data transmitted is usually quite low. For this reason, we will use bits instead of using bytes. Additionally,the purpose of exporting the duration of the flow will be to compute metrics that do not depend on the duration of the flow. In other words, we can not compare the total number of bits and packets between different flows if their duration is completely different, so for that reason, the best solution is to use bits per second and packets per second instead, and that is why we need the duration of each flow.

The previous information will only be exported for flows using TCP. In addition, in order to complement this information so as to facilitate the detection of mining activity, each flow will be aggregated to its complementary, so instead of having two different flows for a connection between server and client (since *Netflow* is unidirectional), we will only have one with both flows together and two directions, inbound and outbound.

To summarise, the final metrics that we will use to determine if a flow contains *Stra-*

tum activity and, consequently, mining activity, are the followings:

- Inbound Packets/s
- Outbound Packets/s
- Inbound Bits/s
- Outbound Bits/s
- Inbound Bits/Packet
- Outbound Bits/Packet
- (Inbound Packets)/(Outbound Packets)
- (Inbound bits)/(Outbound bits)

8 Data collection

Once that we know which information we want to capture, the next step is the collection of this information.

Before starting with the capture of DNS resolutions and network flows, first of all we have to determine from where we will capture. For that reason, I decided to prepare a virtual machine that will be infected with different samples of malware in order to gather traffic traces and flows. The idea behind this action is to observe how cryptomining malware behaves in real simulated scenarios as well as collecting data for the following analysis and creation of datasets.

8.1 Creation of the environment

Therefore, the first step is the creation of the environment. As it was stated above, I will use a virtual machine, concretely, *Virtual Box* with a *Windows 7* operating system. All this software will be installed on a new partition with *Debian 9*.

When testing and studying malware, a crucial aspect is the security of the computer and the network. This consideration is specially important during this stage of the project, hence multiple protection mechanisms were prepared. First of all, the virtual machine is isolated from the real machine and the network by setting the following configurations:

- The type of network established in the virtual machine is NAT, so no one on the network can notice the virtual machine except for the real machine where is located (this works as a gateway for the virtual machine).
- Folding sharing and clipboard sharing are disabled. USB is also disabled on the virtual machine.
- The VPN *windscribe* is used. This avoids that the attacker knows my IP address

or my network, evading future possible attacks.

Once all these previous mechanisms are correctly set, the defenses of the virtual machine such as the firewall or windows defender are disabled, in order to facilitate the infection.

8.2 Obtention of malware samples

After correctly setting the environment, the next stage consists in obtaining malware samples that will be utilised to infect the virtual machine. The needed malware is only the related with cryptomining, that is cryptomining malware or droppers downloading cryptomining malware.

The main source for finding these samples was *URLhaus* [32], a big database containing many online malware samples from different types.

A total of 30 samples were obtained here. Nevertheless, before proceeding with the infection, those samples were examined. In order to correctly classify the samples as malware and, specifically, as cryptomining related malware, two filters were applied:

1. **Malware:** A sample is considered malware if at least 10 AV engines in *VirusTotal* label it as malware, or if there exists at least one report (hybrid analysis or static analysis) stating that it is indeed malware.

2. **Cryptomining related malware:** A sample is considered to be related to cryptomining if at least 10 AV engines in *VirusTotal* label it as *CoinMiner* or some variant, or if there exists at least one report (hybrid or static analysis) stating that is is a miner or a dropper downloading a miner process.

After applying this filter to the collected samples, a total of 20 samples were left. This samples will be used to collect the required data.

8.3 Gathering of data

With the previous environment and malware samples, the virtual machine was infected. The procedure consisted of a set of steps applied for each sample:

1. Recover to a previous state where the virtual machine was not infected.
2. Infect the virtual machine with a single sample.
3. Collect traffic for 10 minutes in form of pcap files using *tcpdump*.
4. Generate nfcapd files from the previous pcap file using *nfcapd* and *softflowd*.
5. Generate a csv file from reading the previous nfcapd using *tcpdump*. This file contains all the previously presented metrics obtained from *Netflow*.

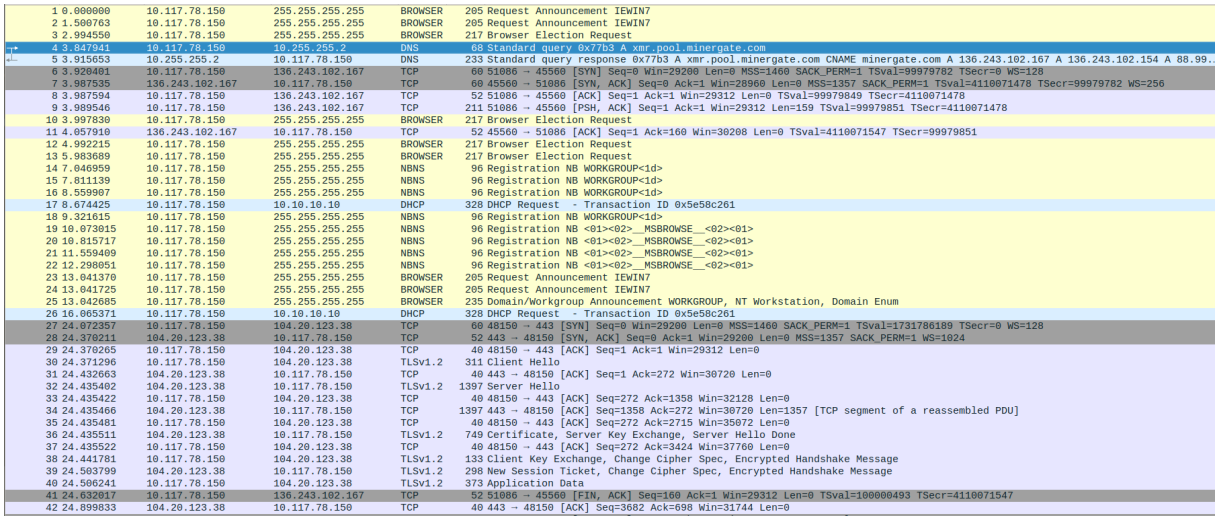
From repeating the previous cycle for each sample, a pcap file and a csv file containing the *Netflow* metrics are obtained for each sample.

It is important to clarify that the pcap file will be the equivalent to the *dnstop* file, since the pcap file was also required for the analysis as well as for the generation of flow. For that reson, for the generation of the dataset the pcap will work as a *dnstop* file due to the fact that only the DNS resolutions will be considered.

9 Data analysis

In the previous section, the data was correctly obtained. The main purpose of this previous data was to analyse it and detect patterns and check behaviours.

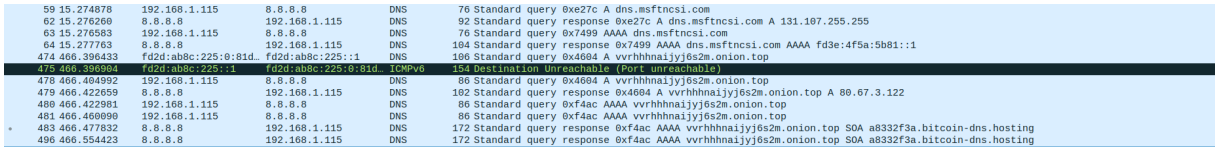
For this reason, the first behaviour that I checked was the connection to the POOLs. As it was stated previously, one of the reasons for checking DNS resolutions was to check if the URLs involved were related to mining pools, since every mining process has to connect to some pool. In figure 4 it can be seen a DNS resolution to a miner pool made by a cryptomining malware, followed by a TCP connection to that pool. This proves that the behaviour was the one expected. After infecting the machine, the mining process starts a new communication with the pool using a standard DNS resolution.



1	0.000000	10.117.78.150	255.255.255.255	BROWSER	205	Request Announcement IEWIN7
2	1.500763	10.117.78.150	255.255.255.255	BROWSER	205	Request Announcement IEWIN7
3	2.994550	10.117.78.150	255.255.255.255	BROWSER	217	Browser Election Request
4	3.827041	10.117.78.150	10.255.255.2	DNS	68	Standard query 0x77b3 A xmr.pool.minergate.com
5	3.915653	10.255.255.2	10.117.78.150	DNS	233	Standard query response 0x77b3 A xmr.pool.minergate.com CNAME minergate.com A 136.243.102.167 A 136.243.102.154 A 88.99..
6	3.920401	10.117.78.150	136.243.102.167	TCP	60	51086 → 45560 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=99979782 TSecr=0 WS=128
7	3.987535	136.243.102.167	10.117.78.150	TCP	60	45560 → 51086 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1357 SACK_PERM=1 TSval=4110071478 TSecr=99979782 WS=256
8	3.987594	10.117.78.150	136.243.102.167	TCP	52	51086 → 45560 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=99979849 TSecr=4110071478
9	3.989546	10.117.78.150	136.243.102.167	TCP	211	51086 → 45560 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=159 TSval=99979851 TSecr=4110071478
10	3.997830	10.117.78.150	255.255.255.255	BROWSER	217	Browser Election Request
11	4.057910	136.243.102.167	10.117.78.150	TCP	52	45560 → 51086 [ACK] Seq=1 Ack=160 Win=30208 Len=0 TSval=4110071547 TSecr=99979851
12	4.992215	10.117.78.150	255.255.255.255	BROWSER	217	Browser Election Request
13	5.983689	10.117.78.150	255.255.255.255	BROWSER	217	Browser Election Request
14	7.046959	10.117.78.150	255.255.255.255	NBNS	96	Registration NB WORKGROUP<1d>
15	7.811139	10.117.78.150	255.255.255.255	NBNS	96	Registration NB WORKGROUP<1d>
16	8.559907	10.117.78.150	255.255.255.255	NBNS	96	Registration NB WORKGROUP<1d>
17	8.674425	10.117.78.150	10.10.10.10	DHCP	328	DHCP Request - Transaction ID 0x5e58c261
18	9.321615	10.117.78.150	255.255.255.255	NBNS	96	Registration NB WORKGROUP<1d>
19	10.073015	10.117.78.150	255.255.255.255	NBNS	96	Registration NB <01><02>__MSBROWSE__<02><01>
20	10.815717	10.117.78.150	255.255.255.255	NBNS	96	Registration NB <01><02>__MSBROWSE__<02><01>
21	11.559409	10.117.78.150	255.255.255.255	NBNS	96	Registration NB <01><02>__MSBROWSE__<02><01>
22	12.298051	10.117.78.150	255.255.255.255	NBNS	96	Registration NB <01><02>__MSBROWSE__<02><01>
23	13.041370	10.117.78.150	255.255.255.255	BROWSER	205	Request Announcement IEWIN7
24	13.041725	10.117.78.150	255.255.255.255	BROWSER	205	Request Announcement IEWIN7
25	13.042685	10.117.78.150	255.255.255.255	BROWSER	235	Domain/Workgroup Announcement WORKGROUP, NT Workstation, Domain Enum
26	16.065371	10.117.78.150	10.10.10.10	DHCP	328	DHCP Request - Transaction ID 0x5e58c261
27	24.072357	10.117.78.150	104.20.123.38	TCP	60	48150 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1731786189 TSecr=0 WS=128
28	24.370211	104.20.123.38	10.117.78.150	TCP	52	443 → 48150 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1357 SACK_PERM=1 WS=1024
29	24.370265	10.117.78.150	104.20.123.38	TCP	40	48150 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0
30	24.371296	10.117.78.150	104.20.123.38	TLSv1.2	311	Client Hello
31	24.432663	104.20.123.38	10.117.78.150	TCP	40	443 → 48150 [ACK] Seq=1 Ack=272 Win=30720 Len=0
32	24.435402	104.20.123.38	10.117.78.150	TLSv1.2	1397	Server Hello
33	24.435422	10.117.78.150	104.20.123.38	TCP	40	48150 → 443 [ACK] Seq=272 Ack=1358 Win=32128 Len=0
34	24.435466	104.20.123.38	10.117.78.150	TCP	1397	443 → 48150 [ACK] Seq=1358 Ack=272 Win=30720 Len=1357 [TCP segment of a reassembled PDU]
35	24.435481	10.117.78.150	104.20.123.38	TCP	40	48150 → 443 [ACK] Seq=272 Ack=2715 Win=35072 Len=0
36	24.435511	104.20.123.38	10.117.78.150	TLSv1.2	749	Certificate, Server Key Exchange, Server Hello Done
37	24.435522	10.117.78.150	104.20.123.38	TCP	40	48150 → 443 [ACK] Seq=272 Ack=3424 Win=37760 Len=0
38	24.441781	10.117.78.150	104.20.123.38	TLSv1.2	133	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
39	24.503799	104.20.123.38	10.117.78.150	TLSv1.2	298	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
40	24.506241	10.117.78.150	104.20.123.38	TLSv1.2	373	Application Data
41	24.632017	10.117.78.150	136.243.102.167	TCP	52	51086 → 45560 [FIN, ACK] Seq=160 Ack=1 Win=29312 Len=0 TSval=100000493 TSecr=4110071547
42	24.899833	104.20.123.38	10.117.78.150	TCP	40	443 → 48150 [ACK] Seq=3682 Ack=698 Win=31744 Len=0

Figure 4: Trace of traffic created by malware

The previous pattern was followed by a majority of the samples. However, there were cases particularly interesting, where the connections to the pools were not so evident. In those cases, the connections were made through proxies. For instance, in figure 5 is display one of these unusual cases, where the mining process was possibly made through onion network.

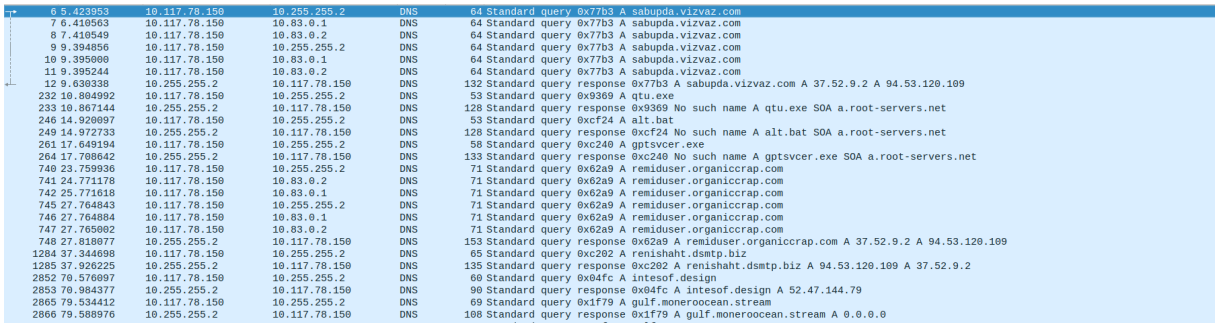


59	15.274878	192.168.1.115	8.8.8.8	DNS	76	Standard query 0xe27c A dns.msftncsi.com
62	15.276260	8.8.8.8	192.168.1.115	DNS	92	Standard query response 0xe27c A dns.msftncsi.com A 131.107.255.255
63	15.276583	192.168.1.115	8.8.8.8	DNS	76	Standard query 0x7499 AAAA dns.msftncsi.com
64	15.277763	8.8.8.8	192.168.1.115	DNS	104	Standard query response 0x7499 AAAA dns.msftncsi.com AAAA fd3e:4f5a:5b81::1
474	466.396433	fd2d:ab8c:225:0:81d…	fd2d:ab8c:225::1	DNS	106	Standard query 0x4604 A vvrhhhnaijyj6s2m.onion.top
475	466.396904	fd2d:ab8c:225::1	fd2d:ab8c:225:0:81d…	ICMPv6	154	Destination Unreachable (Port unreachable)
478	466.404992	192.168.1.115	8.8.8.8	DNS	86	Standard query 0x4604 A vvrhhhnaijyj6s2m.onion.top
479	466.422659	8.8.8.8	192.168.1.115	DNS	102	Standard query response 0x4604 A vvrhhhnaijyj6s2m.onion.top A 80.67.3.122
480	466.422981	192.168.1.115	8.8.8.8	DNS	86	Standard query 0xf4ac AAAA vvrhhhnaijyj6s2m.onion.top
481	466.460090	192.168.1.115	8.8.8.8	DNS	86	Standard query 0xf4ac AAAA vvrhhhnaijyj6s2m.onion.top
483	466.477832	8.8.8.8	192.168.1.115	DNS	172	Standard query response 0xf4ac AAAA vvrhhhnaijyj6s2m.onion.top SOA a8332f3a.bitcoin-dns.hosting
496	466.554423	8.8.8.8	192.168.1.115	DNS	172	Standard query response 0xf4ac AAAA vvrhhhnaijyj6s2m.onion.top SOA a8332f3a.bitcoin-dns.hosting

Figure 5: Possible DNS traffic of a mining program through onion network

The identification of these previous unusual cases strengthen the idea suggested in this project, just relying on only one feature for the detection of cryptomining malware is not a good idea, since in many cases it can fail.

The next analysis consisted in checking how cryptomining malware works when is distributed through other malware. That is checking the behaviour of *droppers* downloading miners. Figure 6 shows clearly the traffic generated by a dropper. As it can be noticed, the dropper tries to connect to several malicious URLs in order to keep downloading malware. Finally, one of the last connections is the one made to the pool *moneroocean*, consequence of a miner downloaded by the dropper.



6	5.423953	10.117.78.150	10.255.255.2	DNS	64	Standard query 0x77b3 A sabupda.vizvaz.com
7	6.410563	10.117.78.150	10.83.0.1	DNS	64	Standard query 0x77b3 A sabupda.vizvaz.com
8	7.410549	10.117.78.150	10.83.0.2	DNS	64	Standard query 0x77b3 A sabupda.vizvaz.com
9	9.394856	10.117.78.150	10.255.255.2	DNS	64	Standard query 0x77b3 A sabupda.vizvaz.com
10	9.395000	10.117.78.150	10.83.0.1	DNS	64	Standard query 0x77b3 A sabupda.vizvaz.com
11	9.395244	10.117.78.150	10.83.0.2	DNS	64	Standard query 0x77b3 A sabupda.vizvaz.com
12	9.630338	10.255.255.2	10.117.78.150	DNS	132	Standard query response 0x77b3 A sabupda.vizvaz.com A 37.52.9.2 A 94.53.120.109
232	10.804992	10.117.78.150	10.255.255.2	DNS	53	Standard query 0x9369 A qtu.exe
233	10.867144	10.255.255.2	10.117.78.150	DNS	128	Standard query response 0x9369 No such name A qtu.exe SOA a.root-servers.net
246	14.920097	10.117.78.150	10.255.255.2	DNS	53	Standard query 0xcf24 A alt.bat
249	14.972733	10.255.255.2	10.117.78.150	DNS	128	Standard query response 0xcf24 No such name A alt.bat SOA a.root-servers.net
261	17.649194	10.117.78.150	10.255.255.2	DNS	58	Standard query 0xc240 A gptsvcer.exe
264	17.708642	10.255.255.2	10.117.78.150	DNS	133	Standard query response 0xc240 No such name A gptsvcer.exe SOA a.root-servers.net
740	23.759936	10.117.78.150	10.255.255.2	DNS	71	Standard query 0x62a9 A remiduser.organiccrap.com
741	24.771178	10.117.78.150	10.83.0.2	DNS	71	Standard query 0x62a9 A remiduser.organiccrap.com
742	25.771618	10.117.78.150	10.83.0.1	DNS	71	Standard query 0x62a9 A remiduser.organiccrap.com
745	27.764843	10.117.78.150	10.255.255.2	DNS	71	Standard query 0x62a9 A remiduser.organiccrap.com
746	27.764884	10.117.78.150	10.83.0.1	DNS	71	Standard query 0x62a9 A remiduser.organiccrap.com
747	27.765002	10.117.78.150	10.83.0.2	DNS	71	Standard query 0x62a9 A remiduser.organiccrap.com
748	27.818077	10.255.255.2	10.117.78.150	DNS	153	Standard query response 0x62a9 A remiduser.organiccrap.com A 37.52.9.2 A 94.53.120.109
1284	37.344698	10.117.78.150	10.255.255.2	DNS	65	Standard query 0xc202 A renishaht.dsmtp.biz
1285	37.926225	10.255.255.2	10.117.78.150	DNS	135	Standard query response 0xc202 A renishaht.dsmtp.biz A 94.53.120.109 A 37.52.9.2
2852	70.576097	10.117.78.150	10.255.255.2	DNS	60	Standard query 0x04fc A intesof.design
2853	70.984377	10.255.255.2	10.117.78.150	DNS	90	Standard query response 0x04fc A intesof.design A 52.47.144.79
2865	79.534412	10.117.78.150	10.255.255.2	DNS	69	Standard query 0x1f79 A gulf.moneroocean.stream
2866	79.588976	10.255.255.2	10.117.78.150	DNS	108	Standard query response 0x1f79 A gulf.moneroocean.stream A 0.0.0.0

Figure 6: Example of DNS traffic generated by a dropper

This previous analysis indicates that *droppers* can be used for the final purpose of mining. Hence, the suggested URL method would be very useful to detect the connection to malicios URLs that may download miners.

Finally, the last analysis to be done was the one regarding the flow. In order to completely understand the patterns that the flow follows, before starting with its analysis, the first step was to gather more flows. For that reason and due to the fact that no more

samples of coin miners were available on the data base, the flows proceeding from public data sets were aggregated to the flows extracted from the samples. Those datasets consisted mainly of pcap files containing both malware and non-malware activity, from which the flows were extracted using the same tools than the ones used with the pcaps files obtained during the capture of data.

The previous datasets were obtained from *Stratosphere Research Laboratory* [33], and they have been used in several papers [34] [35] as well as in some conferences [36]. Concretely, the external files obtained were *2018-02-23_win13*, *2018-03-27_win4*, *2018-04-04_win16*,*2013-12-17_capture1*, *2017_04_30_normal*, *2017-04-18_win-normal*, *2017-04-19_win-normal*, *2017-04-25_win-normal*, *2017-04-28_normal*, *2017-05-01_normal*, *2017-05-02_normal*, *2017-07-03_capture_win2*. Some of these files contain malware activity, others contain non-malware traffic. The usage of these data sets will be helpful in order to have non-malware traffic so as to correctly identify patterns.

Once that a total dataset of flow metrics is created, each flow is labeled with *yes* or *no*, indicating if the flow is using *Stratum* or not. In order to correctly classify each flow, a deep packet inspection is performed, since the pcap files are available. If the content of each flow shows a JSON method containing information about mining, then the flow is labeled as *yes*. From this point, the process continued by checking that the traffic can be correctly identified. The most important feature was the low quantity of information transmitted along with the fact that the client sends very small quantity of data whereas the server sends the data in a steady way. In order to check the existing patters, three plots were created. Every plot shows the flows from *Stratum* connections (the ones in black) and from non Stratum, *i.e.* non mining, connection. In the first one, Figure 7, we can see the behaviour of the inbound and outbound bits per second. As it was predicted, the number of bits per second sent by the client on a *Statrum* connection (BitsOut) is very small, while the number of bits per second received by the client is bigger (for the above stated reason that the server is sending constantly), but is still small compare with normal traffic.

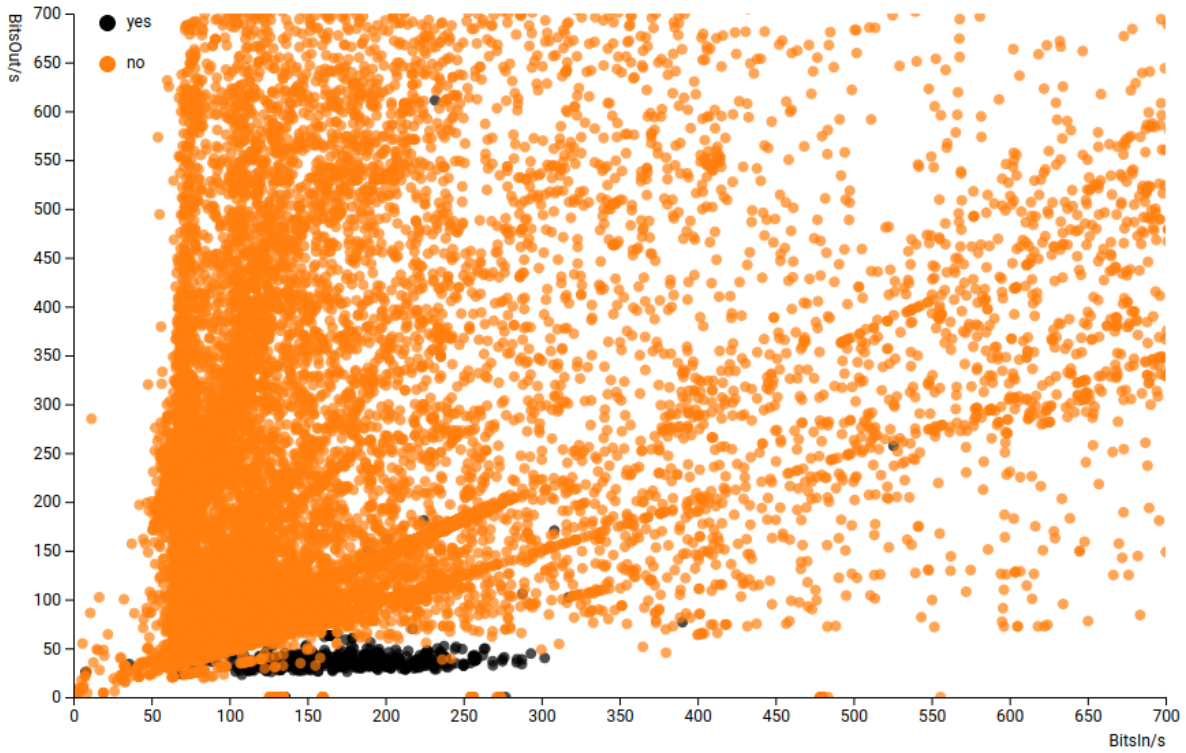


Figure 7: Patterns of inbound and outbound bits per second

In Figure 8, the same conclusions are obtained here. Server sends more packets per second than the client, even though the difference is less significant than in the case of bits. However, still the quantity of data transmitted (in this case packets) in mining connections is still much smaller than in case of other traffic.

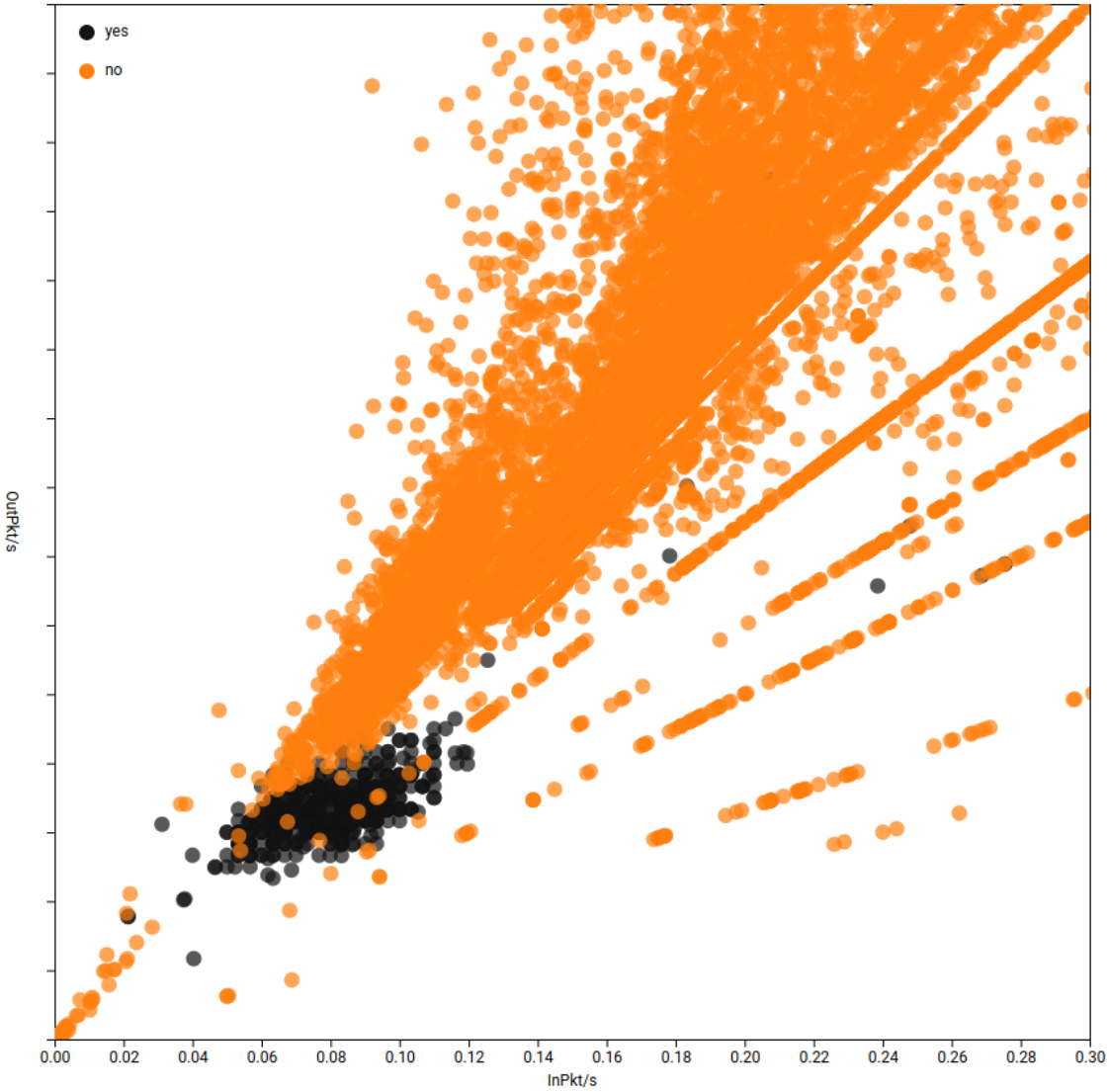


Figure 8: Patterns of inbound and outbound packets per second

In the case of Figure 9, the identification is not as clear as in the other cases. Here the quantity of bits per packet transmitted by the client and the server in non mining connections can be very reduced too in some cases. However, the mining flows are grouped in a very specific area, quite separated from non *Stratum* traffic. Still the number of outbound bits per second is bigger than the inbound, due to the fact that the quantity of bits transmitted is much bigger in case of the server, while the number of packets sent in both cases only differs slightly.

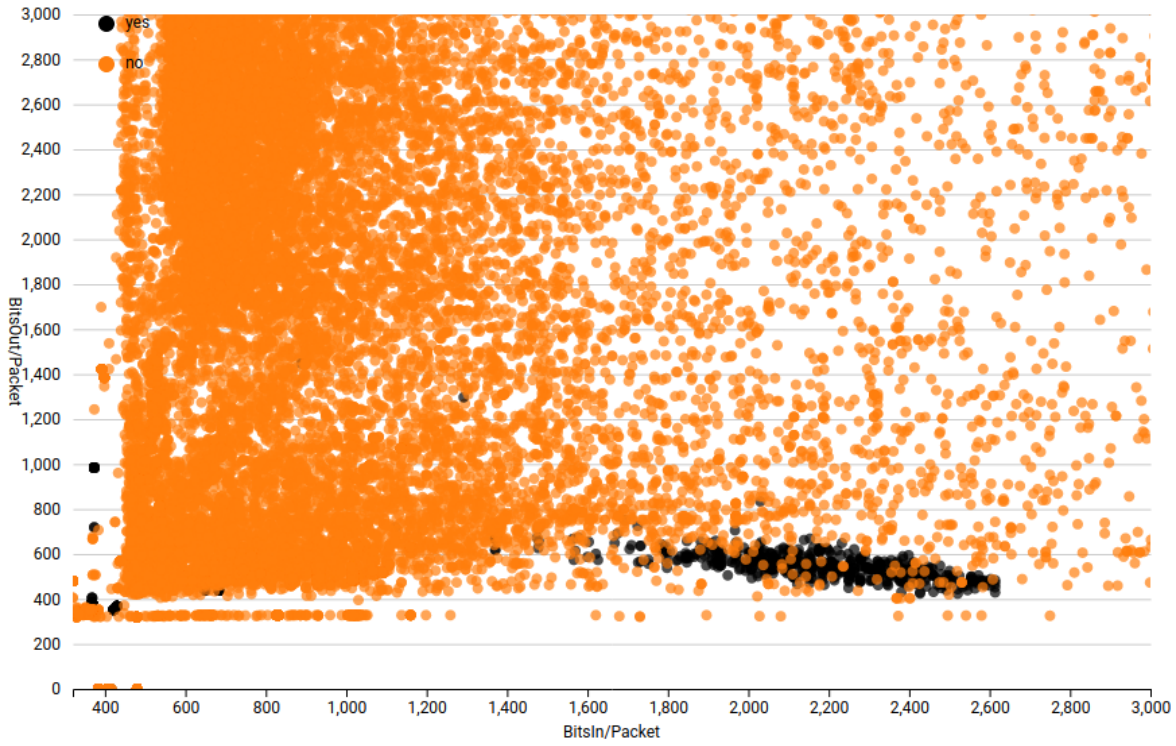


Figure 9: Patterns of inbound and outbound bits per packet

From the previous analysis, the conclusion is clear, there are patterns that can be distinguished by a machine learning algorithm in order to classify flows into mining related flows or non mining related flows. However, as it was already stated, these patterns are only seen in cases of *Stratum* or related protocol. If the connection protocol shows different patterns regarding the quantity of transmitted data, then a new analysis would be required in order to check possible patterns.

10 Machine Learning algorithms

In this section, the development of the machine learning algorithms as well as the creation of their required datasets are explained. Furthermore, a comparison between the different models to consider for each method is shown.

10.1 Datasets

From the previous section, one of the datasets is already set, which is the dataset with netflow metrics. This dataset is already built. In table 8 the total number of flows can be observed. Although is not a big data set, I consider that it will be enough to train the model if the correct parameters are selected.

	Flow
yes	**10,184**
no	**46,762**
Total	**56,946**

Table 8: Number of flows labeled as mining (yes) and non mining

Regarding the dataset containing the URLs for the *Malicious URL method*, the first part of the creation of this dataset consisted in gathering all the URLs from the DNS resolutions in the samples and external pcap files obtained from public repository. Afterwards, this URLs are filtered using the parameters presented before. That is checking that at least 10 AV in *Virus Total* detect it as malicious, or if there exists at least one report stating that the URL is malicious.

The second part of the dataset construction consisted in joining aggregating data to the previous data. This was done by obtaining public lists of Phishing URLs [37], lists of URLs dedicated to SPAM [38], URLs with malware [39], and finally clean URLs [40].

Finally, from all the previous data, we have now a dataset with either a label "bad"

meaning that the URL is malicious, or "good" if the URL is considered benign. In table 9 the total number of URLs per type is shown, as well as the total.

	URL
good	**1,344,205**
bad	**109,118**
Total	**1,453,323**

Table 9: Number of malicious and benign URLs

Finally, regarding the dataset for the *Pool method*, all the URLs involved in a DNS resolution in some of the pcap files obtained from the malware samples (and from the public repository of datasets) are checked with a list of all the current living pools [41]. If the selected URL is a pool, it is labeled with a *1* (indicating that it is a pool indeed), otherwise is labeled with a *0*. At the end of the process, all the pools from the list of living pools that have not appeared yet are introduced into the dataset. The other non pool related URLs are obtained from the data set of the *Malicious URL method*. Of course these URLs are not from pools, so they are labeled as *0*. In table 9 it can be noticed both the number of URLs from mining pools and the number of URLs not from mining pools.

	Pools
1	**2,031**
0	**218,860**
Total	**220,891**

Table 10: Number of URLs from mining pools and from non mining pools

10.2 Malicious URL Method

With the dataset prepared, the next step is to develop a machine learning algorithm which receives an URL and determines whether is malicious or not. The quickest way to do is is just by inspecting the URL, without executing any code from the URL, so the

prediction should be done just using the lexical information appearing in the URL. In other words, the idea is to apply a *static analysis*.

In order to do the above stated task, one way (and the one explored in this project) is to split each URL into parts also known as features. For instance, from the URL *cassiope.cz/js/bin.exe* we can obtain the following vector of features: *cassiope*, *cz*, *js*, *bin*, *exe*.

Now, we can apply the same idea to all the malicious URLs, so we will have a vector of vectors of features. We can now combine all of them, since we want to avoid repetitions, generating a vector of features. So now we have several URLs, where the total of URLs is denoted by U in order to formalise the problem, and a vector of features or words, where the total of words is identified by W.

The term frequency, which is the number of occurrences of a word inside an URL, can be denoted by $TF(u, w) = f_u, w$, where $u \in U$, $w \in W$, and f_u, w is just the number of occurrences of word w in URL u. This idea is useful to extract a more abstract concept, which is to know how strange or common is a word in all the entire set of malicious URLs (the same will be then applied for benign URLs). Inverse Document Frequency (IDF) is perfect for this purpose. The IDF is computed by: $IDF(w, U) = log\frac{|U|}{|\{u \in U : t \in u\}|}$

If we apply the previous value to every word in the vector of features extracted from the list of malicious URLs, finally we well have how rare and common is each word. If a word has a small IDF, it means that this word is quite common among malicious. How much bigger the value, much rare is the word.

However, we cannot use that to fit our model yet, since the differences between common and rare words are not very significant, or at least not enough to ensure a good classification. We should guarantee that common cases are filtered, so for that reason I applied weights. Term frequency-Inverse document frequency (TFIDF) perfectly defines this: $TFIDF(w, u, U) = TF(w, u) * IDF(w, U)$.

In the TFIDF we will get higher scores in case of common words in the given URL but rare through the set of URLs. Now this scores work well to train the models.

The main idea is, from just an URL, which is a set of characters containing first a protocol, and then a resource, extract statistical features so a machine learning algorithm can be fitted. The features must be based on lexical features, since I want to apply the idea of the *static analysis* of malware to the URLs, so no access or execution has to be done.

So finally, the entire formalisation of the problem is shown afterwards. Given a dataset $\{(u_1, y_1), ..., (u_n, y_n)\}$ where $U = \{u_1, ..., u_n\}$, $n \geq 2$, $y_i \in \{good, bad\}$ and $1 \leq i \geq n$.

1. Extract features: $u_i \rightarrow x_i$ where $x_i \in \mathbb{R}^d$, being this a d-dimensional feature vector. $\forall v \in x_i, TFIDF(v, u_i, U) = z_i$. In other words, z_i is the TFIDF score of a feature inside the vector of features obtained from the url u_i.

2. Predict: $\forall z$ apply $g : \mathbb{R}^d \rightarrow \{good, bad\}$

With the solution proposed, now it is time to apply the algorithm and train different models in order to realise which model obtains the best results. Due to the big size of the the dataset, only two methods were feasible enough to use: *Multinomial Naive Bayes*, *Logistic Regression*.

- **Multinomial Naive Bayes**: This model works well for huge datasets due to the fact that is very simple to construct. The main idea is to use the dataset in order to create a score, in a way that greater scores are associated with one class, and smaller scores are associated with the other, so basically the idea is to compare a score with this threshold. The score is obtained from the input variables [42]. Multinomial extension indicates that the probabilities required follow a multinomial distribution.

- **Logistic Regression**: For each given entry, this model predicts the probability that it belongs to one of the categories, using a regression model previously built. A

decision threshold should be set in order to use this method as a classifier.

10.2.1 Comparison and results

The procedure followed mainly consisted in executing an exhaustive grid search for each model so as to find which combination of values performed best. In other words, the idea was to try several values for each parameter of each model. Once that the best result is obtained for each method, the best from one method is compared with the best of the other. Finally, the best one is the chosen one.

In the case of *Multinomial Naive Bayes*, the best combination of parameters using python's module *sklearn* was: *alpha=0.1*, *fit_prior=True*. On the contrary, in the case of *Logistic Regression*, the best combination was: *class_weight=balanced*, *fit_intercept=False*, *max_iter=100*, *multi_class=multinomial*, *solver=newton-cg*.

Before starting the comparison, the first step is to check the metrics that will be used. First of all, we have *accuracy*, which can be defined as: $Accuracy = \frac{CorrectPredictions}{WrongPredictions}$. The next metric that we have is Precision, which proposes an answer to the question: How many of the positives predictions were right?. The formula is: $Precision = \frac{TP}{TP + FP}$. Finally, the last indicator is Recall, which answers the question: How many of the real positives were correctly identified?. This is the formula: $Recall = \frac{TP}{TP + FN}$. With these concepts clear, in table 10 all of them are compared.

Model	Accuracy	Precision	Recall
Multinomial Naive Bayes	0.987480	0.991762	0.994719
Logistic Regression	0.985733	0.991895	0.992688

Table 11: Comparison between different models for *Malicious URL method*

Additionally to the previous metrics, in table 12 and 13 can be found the confusion matrices for both of the previous models.

Multinomial Naive Bayes	Real Good	Real Bad
Predicted Good	267,278	2,220
Predicted Bad	1,419	19,748

Table 12: Confusion matrix for Multinomial Naive Bayes for *Malicious URL Method*

Logistic Regression	Real Good	Real Bad
Predicted Good	266,911	2,181
Predicted Bad	1,966	19,607

Table 13: Confusion matrix for Logistic Regression for *Malicious URL Method*

From the previous results, the best chosen model was **Multinomial Naive Bayes**, due to the fact that *Multinomial Naive Bayes* obtains less FN (understanding negative as Bad), even though all the other metrics are quite similar in both cases.

10.3 Pool Method

For this method, I applied exactly the same idea than the one applied in the previous method. The reason is simple, instead of trying to use the URLs to detect malware patterns, we can use the URLs to detect pools patterns, which are identified by *1*.

So again, the entire formalisation of the problem is shown here. Given a dataset $\{(u_1, y_1), ..., (u_n, y_n)\}$ where $U = \{u_1, ..., u_n\}$, $n \geq 2$, $y_i \in \{1, 0\}$ and $1 \leq i \geq n$.

1. Extract features: $u_i \rightarrow x_i$ where $x_i \in \mathbb{R}^d$, being this a d-dimensional feature vector. $\forall v \in x_i, TFIDF(v, u_i, U) = z_i$. In other words, z_i is the TFIDF score of a feature inside the vector of features obtained from the url u_i.

2. Predict: $\forall z$ apply $g : \mathbb{R}^d \rightarrow \{1, 0\}$

With this solution, again different models are trained in order to find the best one.

Due to the fact that now the dataset is much smaller than the previous one, more methods were studied: *Multinomial Naive Bayes*, *Decision Tree Classifier*, *K Neighbors Classifier*, *Logistic Regression*, *Random Forest Classifier*.

- **Decision Tree Classifier**: The idea behind this model is simple, just build a tree where each node is a condition and the each leave is one of the classes.For every input, a path is followed until one leaf is reached [42].
- **K Neighbors Classifier**: The idea of KN neighbors is that similar objects are near to each other. In other words, the algorithm computes the distance, *i.e.* the difference between each new entry and a group of neighbours. Locate it to the closest group, *i.e.* more probably category.
- **Random Forest Classifier**: This model consists of a significant number of decision trees. Each entry is executed on each tree, so each decision tree in the random forest generates an output (a class prediction). The class mostly predicted is the final output.

10.3.1 Comparison and results

The procedure is the same than the one from the previous method.

The best combination of parameters for each model using python's module *sklearn* was:

- **Multinomial Naive Bayes**: *alpha=0.05*, *fit_prior=True*.
- **Decision Tree Classifier**:
  - *criterion=gini*
  - *splitter=random*
  - *max_depth=None*

  - *min_samples_split=2*
  - *min_samples_leaf=1,*
  - *min_weight_fraction_leaf=0.0*
  - *max_features=None*
  - *random_state=None*
  - *max_leaf_nodes=None*
  - *min_impurity_decrease=0.0*
  - *min_impurity_split=None*
  - *class_weight=None*
  - *presort=deprecated*
  - *ccp_alpha=0.0.*

- **K Neighbors Classifier**:
  - *n_neighbors=5*
  - *weights=uniform*
  - *algorithm=auto*
  - *leaf_size=30*
  - *p=2*
  - *metric=mikowski*
  - *metric_params=None*

- **Logistic Regression**:

  - *class_weight=balanced*
  - *fit_intercept=True*
  - *max_iter=100*
  - *multi_class=ovr*
  - *solver=newton-cg*

- **Random Forest Classifier**:
  - *n_estimators=100*
  - *class_weight=balanced_subsample*
  - *criterion=gini*
  - *max_depth=None*
  - *min_samples_split=2*
  - *min_samples_leaf=15*
  - *min_weight_fraction_leaf=0.0*
  - *max_features=auto,*
  - *max_leaf_nodes=None*
  - *min_impurity_decrease=0.0*
  - *min_impurity_split=None*
  - *bootstrap=True*
  - *oob_score=True*
  - *random_state=None*

In table 14, the comparison can be noticed.

Model	Accuracy	Precision	Recall
Multinomial Naive Bayes	0.998778	0.967391	0.894472
Decision Tree Classifier	0.998800	0.965333	0.900498
K Neighbors Classifier	0.997985	0.943343	0.828358
Logistic Regression	0.998914	0.953086	0.930120
Random Forest Classifier	0.983499	0.139706	0.131034

Table 14: Comparison between different models for *POOL method*

As in the previous method, confusion matrices are shown in tables 15 to 19.

Multinomial Naive Bayes	**Real 1**	**Real 0**
Predicted 1	356	12
Predicted 0	42	43,769

Table 15: Confusion matrix for Multinomial Naive Bayes for *POOL Method*

Decision Tree Classifier	**Real 1**	**Real 0**
Predicted 1	362	13
Predicted 0	40	43,764

Table 16: Confusion matrix for Decision Tree Classifier for *POOL Method*

K Neighbors Classifier	**Real 1**	**Real 0**
Predicted 1	333	20
Predicted 0	69	43,757

Table 17: Confusion matrix for K Neighbors Classifier for *POOL Method*

Logistic Regression	Real 1	Real 0
Predicted 1	386	19
Predicted 0	29	43,745

Table 18: Confusion matrix for Logistic Regression for *POOL Method*

Random Forest Classifier	Real 1	Real 0
Predicted 1	57	351
Predicted 0	378	43,393

Table 19: Confusion matrix for Random Forest Classifier for *POOL Method*

From the above results, the best option to choose is **Logistic Regression**, since it achieves a very good accuracy, precision and recall.

10.4 Flow Method

The idea of this algorithm is to detect the patters noticed on the analysis section, using metrics related to the quantity of data transmitted. For this purpose, the models trained are: *Multinomial Naive Bayes*, *Decision Tree Classifier*, *K Neighbors Classifier*, *Logistic Regression*, *Random Forest Classifier*. All of them were explained above.

10.4.1 Comparison and results

One last time, the same procedure is applied now, so the best combination of parameters for each model using again python's module *sklearn* was:

- **Multinomial Naive Bayes**: *alpha=0.25, fit_prior=True*.
- **Decision Tree Classifier**:

  - *criterion=entropy*
  - *splitter=best*
  - *max_depth=None*
  - *min_samples_split=10*
  - *min_samples_leaf=3,*
  - *min_weight_fraction_leaf=0.0*
  - *max_features=None*
  - *random_state=None*
  - *max_leaf_nodes=None*
  - *min_impurity_decrease=0.0*
  - *min_impurity_split=None*
  - *class_weight=None*
  - *presort=deprecated*
  - *ccp_alpha=0.0.*

- **K Neighbors Classifier**:
  - *n_neighbors=2*
  - *weights=uniform*
  - *algorithm=auto*
  - *leaf_size=10*
  - *p=3*

  - *metric=distance*
  - *metric_params=None*

- **Logistic Regression**:
  - *class_weight=balanced*
  - *fit_intercept=True*
  - *max_iter=100*
  - *multi_class=multinomial*
  - *solver=nlbfgs*

- **Random Forest Classifier**:
  - *n_estimators=100*
  - class_weight=balanced
  - *criterion=gini*
  - *max_depth=None*
  - *min_samples_split=2*
  - *min_samples_leaf=15*
  - *min_weight_fraction_leaf=0.0*
  - *max_features=auto,*
  - *max_leaf_nodes=None*
  - *min_impurity_decrease=0.0*
  - *min_impurity_split=None*

- *bootstrap=True*
- *oob_score=True*
- *random_state=None*

In table 20 the comparison between models is performed.

Model	Accuracy	Precision	Recall
Multinomial Naive Bayes	0.613521	0.989972	0.536437
Decision Tree Classifier	0.998683	0.999149	0.999255
K Neighbors Classifier	0.999122	0.999251	0.999679
Logistic Regression	0.981124	0.991825	0.985150
Random Forest Classifier	0.999473	0.999464	0.999893

Table 20: Comparison between different models for *Flow method*

Additionally to the previous metrics, in tables 21-24, each confusion matrix is shown. Yes means that the flow contains mining activity using *Stratum*.

Multinomial Naive Bayes	**Real yes**	**Real no**
Predicted yes	5,035	51
Predicted no	4,351	1,953

Table 21: Confusion matrix for Multinomial Naive Bayes for *Flow Method*

Decision Tree Classifier	**Real yes**	**Real no**
Predicted yes	9,390	8
Predicted no	7	1,985

Table 22: Confusion matrix for Decision Tree Classifier for *Flow Method*

K Neighbors Classifier	Real yes	Real no
Predicted yes	9,343	7
Predicted no	3	2,037

Table 23: Confusion matrix for K Neighbors Classifier for *Flow Method*

Logistic Regression	Real yes	Real no
Predicted yes	9,221	76
Predicted no	139	1,954

Table 24: Confusion matrix for Logistic Regression for *Flow Method*

Random Forest Classifier	Real yes	Real no
Predicted yes	9,323	5
Predicted no	1	2,061

Table 25: Confusion matrix for Random Forest Classifier for *Flow Method*

Due to the almost perfect values obtained in the confusion matrix as well as in the accuracy, precision and recall, there is no doubt that the best model in that case is the **Random Forest Classifier**.

11 Results

As a summary, the best model for each method is:

- *Malicious URL Method*: **Multinomial Naive Bayes**
- *Pool Method*: **Logistic Regression**
- *Flow Method*: **Random Forest Classifier**

In tables 26, 27, 28 are shown their confusion matrices.

Multinomial Naive Bayes	**Real Good**	**Real Bad**
Predicted Good	267,278	2,220
Predicted Bad	1,419	19,748

Table 26: Confusion matrix for Multinomial Naive Bayes for *Malicious URL Method*

Logistic Regression	**Real 1**	**Real 0**
Predicted 1	386	19
Predicted 0	29	43,745

Table 27: Confusion matrix for Logistic Regression for *POOL Method*

Random Forest Classifier	**Real yes**	**Real no**
Predicted yes	9,323	5
Predicted no	1	2,061

Table 28: Confusion matrix for Random Forest Classifier for *Flow Method*

As it can be noticed in the previous tables, each of the methods chosen performs quite well. Actually, the *Pool Method*, *Netflow Method* obtain very good results. On the

contrary, *Malicious URL Method* even achieving good levels of accuracy, precision and recall, the total number of false positives (predicted good when was bad) as well as false negatives is noticeably in comparison with the other methods.

12 Conclusions

From this project there is a main conclusion that can be extracted, which is that cryptomining malware can be easily identified with the appropriate tools. This was in fact the main goal of this project, the study of this possibility and its development.

The development of the previous idea led me to the definition of three machine learning methods, each one of them based on a very specific feature of this type of malware. Each method performed reasonably well, specially in the case of detecting patterns in flow and detecting connections to pools, but the second main conclusion can be obtained here, since none of these methods should stand alone. The reason is simple, none of them by themselves are enough to ensure that cryptomining malware will be detected. In the case of the detection based on the malicious URLs, this conclusion can be extracted just from the high number of false positives and false negatives. However, this is understandable from the point of view that what defines the malicious intention of an URL is its content. Even though in this project has been proved that the URLs can contain hints to detect malicious content (since the results obtained are not bad), in many cases this hints are not enough to detect the reality behind each URL, so a false prediction is obtained.

For the case of detection using flow metrics, the main problem is when mining malware does not use *Stratum* protocol. Actually, the idea that miners will use *Stratum* is a strong assumption, and if it is not use and the alternative is not related to it, the identification will be compromised.

Finally, in the case of the identification using pools, if the attackers use proxies or private pools, this method will not work. Clearly this is not a general rule, as it is not the usage of a *non-Stratum* protocol, because otherwise the results would have been worse, but the main conclusion that should be clear is that each method is highly susceptible to failure if no others mechanisms are applied. For this reason, working with all of three methods offer a much secure proven alternative.

12.1 Future work

Since the time to develop the project is very limited, many interesting ideas were removed, so in case of having the opportunity, they could be retaken and explored deeper.

One of this ideas involved the identification of different types of cryptocurrencies by the identification of pools and flow metrics. With the union of both methods, a new step after the initial detection of miners could be added.

Another interesting idea would be to modify flow method so as to it could be able to detect *non-Stratum* traffic too. The same idea of method expansion can be also applied to pools method, by studying a mechanism that would allow the identification of connections to proxies with the intention to mine.

13 Bibliography

References

[1] A. Victor, "Introducing cryptocurrency," *READS Capital*, vol. 1, p. 2, October 2017.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, March 2009.

[3] K. Zīle and R. Strazdiņa, "Blockchain use cases and their feasibility," *Applied Computer Systems*, vol. 23, pp. 12–20, May 2018.

[4] M. Crosby, Nachiappan, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," *Applied Innovation Review*, vol. 2, June 2016.

[5] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," June 2017.

[6] J. Z. i. Muñoz, J. Suárez-Varela, and P. Barlet-Ros, "Detecting cryptocurrency miners with netflow/ipfix network measurements," in *2019 IEEE International Symposium on Measurements Networking (M N)*, pp. 1–6, July 2019.

[7] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," January 2019.

[8] Webroot, "Webroot threat report: Mid-year update," September 2018. [Online]. Available: https://perma.cc/3M2Z-Q76Y.

[9] M. Saad, A. Khormali, and A. Mohaisen, "End-to-end analysis of in-browser cryptojacking," September 2018.

[10] J. Liu, Z. Zhao, X. Cui, Z. Wang, and Q. Liu, "A novel approach for detecting browser-based silent miner," in *2018 IEEE Third International Conference on Data*

Science in Cyberspace (DSC), pp. 490–497, June 2018.

[11] A. Swedan, A. Khuffash, O. Othman, and A. Awad, "Detection and prevention of malicious cryptocurrency mining on internet-connected devices," August 2018.

[12] Bitcoin, "Mining guide," 2020. [Online]. Available: https://bitcoin.org/en/mining-guide.

[13] Avast, "Avast web page," 2020. [Online]. Available: https://www.avast.com.

[14] Malwarebytes, "Malwarebytes web page," 2020. [Online]. Available: https://www.malwarebytes.com/.

[15] NioGuard Security Lab, "Anti-cryptojacking test," July 2019. [Online]. Available: https://www.nioguard.com/2019/08/anti-cryptojacking-test-july-2019.html.

[16] R. Keramidas, "No coin," February 2018. [Online]. Available: https://github.com/keraf/NoCoin.

[17] Tunghobrens, "Anti miner–coin minerblock," February 2018. [Online]. Available: https://tinyurl.com/ybf3jcsj.

[18] VirusTotal, "Virustotal," 2020. [Online]. Available: https://www.virustotal.com/.

[19] Hybrid Analysis, "Hybrid analysis," 2020. [Online]. Available: https://www.hybrid-analysis.com/.

[20] Infosec, "Latest analyses," 2020. [Online]. Available: https://infosec.cert-pa.it/analyze/submission.html.

[21] FIB-UPC, "Normativa del treball final de grau del grau en enginyeria informàtica de la FIB," 2012. [Online]. Available: https://www.fib.upc.edu/sites/fib/files/documents/actes/normativatfg-gei-2012-09-26.pdf.

[22] Microsoft, "Download virtual machines," 2017. [Online]. Available: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/.

[23] Payscale, "Average Project Manager, Information Technology (IT) Salary in Spain," 2020. [Online]. Available: https://www.payscale.com/research/ES/Job=Project_Manager%2C_Information_Technology_(IT)/Salary.

[24] Payscale, "Average Systems Administrator Salary in Spain," 2020. [Online]. Available: https://www.payscale.com/research/ES/Job=Systems_Administrator/Salary.

[25] Payscale, "Average Data Scientist Salary in Spain," 2020. [Online]. Available: https://www.payscale.com/research/ES/Job=Data_Scientist/Salary.

[26] Payscale, "Average Machine Learning Engineer Salary in Spain," 2020. [Online]. Available: https://www.payscale.com/research/ES/Job=Machine_Learning_Engineer/Salary.

[27] Microsoft, "Shop windows 10," 2020. [Online]. Available: https://www.microsoft.com/en-us/store/b/windows?icid=CNavSoftwareWindows&activetab=tab:shopwindows10.

[28] Outervision, "Power supply calculator," 2020. [Online]. Available:https://outervision.com/power-supply-calculator.

[29] Kaspersky, "Trojan-dropper," 2020. [Online]. Available: https://encyclopedia.kaspersky.com/knowledge/trojan-dropper/.

[30] Bitcoin, "Stratum nining protocol," 2015. [Online]. Available: https://en.bitcoin.it/wiki/Stratum_mining_protocol.

[31] CISCO, "Netflow," 2020. [Online]. Available: https://www.cisco.com/c/en/us/tech/quality-of-service-qos/netflow/index.html.

[32] URLhaus, "Urlhaus database," 2020. [Online]. Available: https://urlhaus.

abuse.ch/.

[33] S. lab, "Datasets overview," 202. [Online]. Available: https://www.stratosphereips.org/datasets-overview.

[34] S. García, "Survey on network-based botnet detection methods," *Security and Communication Networks*, January 2013.

[35] M. J. Erquiaga, S. García, and C. G. Garino, "Observer effect: How intercepting https traffic forces malware to change their behavior," in *Computer Science – CACIC 2017* (A. E. De Giusti, ed.), (Cham), pp. 272–281, Springer International Publishing, 2018.

[36] M. Rigaki and S. Garcia, "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection," in *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 70–75, May 2018.

[37] OpenPhish, "Openphish - phishing intelligence," 2020. [Online]. Available:https://openphish.com/.

[38] JWSPAMSPY, "Spam domain blacklist," 2020. [Online]. Available:http://www.malwaredomainlist.com/.

[39] M. D. List, "Malware domain list," 2017. [Online]. Available:http://www.malwaredomainlist.com/.

[40] Majestic, "The majestic million," 2020. [Online]. Available:https://majestic.com/reports/majestic-million.

[41] Investoon, "Mining pools live monitoring tools," 2020. [Online]. Available: https://investoon.com/mining_pools/.

[42] X. Wu, V. Kumar, R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. Mclachlan, S. K. A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, December

2007.